4° edizione
preparazione per l'esame
LPIC-1



Amministrare GNU/Linux

Simone Piccardi

Truelite Srl http://www.truelite.it info@truelite.it

Amministrare GNU/Linux
Quarta edizione

Simone Piccardi

Truelite Srl – http://www.truelite.it – info@truelite.it

Amministrare GNU/Linux – Quarta edizione

Questa documentazione libera è stata sviluppata specificamente per le attività formative di **Lici**, *Linux Certification Institute*.

La versione originale di questo manuale, rilasciata sotto licenza GNU FDL, viene distribuita su internet a partire dall'indirizzo:

https://labs.truelite.it/truedoc

dove saranno pubblicate nuove versioni ed aggiornamenti.

LiCI, *Linux Certification Institute*, è un ente di formazione dedicato al mondo del free software; nasce da una collaborazione fra due aziende, Metamarketing Service s.r.l. e Truelite s.r.l.

LiCI offre percorsi formativi personalizzati per aziende, certificazione LPI, e-learning, piattaforma di test per esami, servizi di supporto a strutture formative ed una propria casa editrice con pubblicazioni altamente specializzate.

ISBN 978-1-326-16084-5

Per informazioni:

Lici - Linux Certification Institute
Via Jacopo Nardi 71,
50132 Firenze, Italy
Tel: 800913096
Fax: +39-0552342954
e-mail: mailto:info@lici.it
web: http://www.lici.it



Indice

1 L'architettura di un sistema GNU/Linux **1**
1.1 L'architettura del sistema. 1
1.1.1 L'architettura di base. 1
1.1.2 Il funzionamento del sistema 3
1.1.3 Alcune caratteristiche specifiche di Linux 5
1.2 L'architettura dei file 6
1.2.1 Il *Virtual File System* e le caratteristiche dei file. 6
1.2.2 L'architettura di un filesystem e le proprietà dei file 10
1.2.3 L'albero dei file ed il *Filesystem Hierarchy Standard* 18
1.3 L'architettura dei processi 25
1.3.1 Le caratteristiche dell'architettura dei processi 25
1.3.2 Le proprietà dei processi 26
1.3.3 I segnali 36
1.3.4 La gestione delle priorità 40
1.4 Il controllo degli accessi 41
1.4.1 L'identificazione di utenti e gruppi 41
1.4.2 I permessi dei file 43
1.4.3 La gestione dei permessi dei file 49
1.4.4 Altre operazioni privilegiate 52

2 La shell e i comandi **57**
2.1 L'interfaccia a linea di comando. 57
2.1.1 La filosofia progettuale 57
2.1.2 Sessioni di lavoro e *job control* 60
2.1.3 Sintassi e funzionalità di base della riga di comando 67
2.1.4 La redirezione dell'I/O 83
2.1.5 Scripting elementare 88
2.1.6 Le modalità operative e la configurazione della shell 97
2.2 I comandi dei file 102
2.2.1 Caratteristiche comuni 103
2.2.2 I comandi per le ricerche dei file 105
2.2.3 I comandi per visualizzare il contenuto dei file 110
2.2.4 I comandi di elaborazione dei contenuti dei file 113

2.2.5 Ricerche ed elaborazioni sui file attraverso le espressioni regolari 120
2.3 Gli editor di testo 126
2.3.1 Introduzione 127
2.3.2 Un editor evoluto: `emacs` 128
2.3.3 Un editor di base, `vi` 131
2.3.4 Altri editor 135
2.4 Altri comandi di utilità 138
2.4.1 I comandi di ausilio per la redirezione 138
2.4.2 I comandi per la documentazione 141
2.4.3 I comandi per la gestione dei tempi 145
2.4.4 I comandi per le informazioni sugli utenti e il sistema 148
2.4.5 I comandi per le informazioni diagnostiche 151
2.4.6 I comandi per la verifica di integrità 155

3 La configurazione dei servizi 169
3.1 I file di configurazione 169
3.1.1 Una panoramica generale 169
3.1.2 La gestione delle librerie condivise 170
3.1.3 La gestione dei parametri del kernel con *sysctl* 173
3.1.4 Localizzazione e internazionalizzazione 175
3.2 I servizi di base 180
3.2.1 I servizio di esecuzione periodica 180
3.2.2 Il servizio *at* 184
3.2.3 Il servizio del *syslog* 185
3.2.4 Il sistema di rotazione dei file di log 190
3.3 L'*X Window System* 193
3.3.1 Introduzione a *X Window* 193
3.3.2 La configurazione del server X 195
3.3.3 L'avvio di una sessione di lavoro sotto X 202
3.3.4 L'uso di X Window dal lato client 208
3.3.5 Programmi per l'accessibilità 212
3.4 Il sistema di stampa 213
3.4.1 Introduzione generale 213
3.4.2 Il *Common Unix Printing System* 215
3.4.3 I comandi di gestione per CUPS 219
3.4.4 Il sistema di stampa in stile BSD 223

4 Amministrazione ordinaria del sistema 225
4.1 Archiviazione e backup 225
4.1.1 Criteri generali per il backup 225
4.1.2 I comandi `tar` e `cpio` 230
4.1.3 I comandi `dump` e `restore` 235
4.1.4 Altri comandi per il backup 237
4.2 La gestione dei pacchetti software 240
4.2.1 L'installazione diretta dai sorgenti 240

4.2.2 La gestione dei pacchetti con `rpm` e `yum` 244
4.2.3 Il sistema di gestione dei pacchetti APT 250
4.3 La gestione di utenti e gruppi 258
4.3.1 Una visione generale 258
4.3.2 Il database di utenti e gruppi 260
4.3.3 I comandi per la gestione di utenti e gruppi 264
4.3.4 Impersonare i ruoli di altri utenti e gruppi 270
4.3.5 Le configurazioni della gestione degli utenti e degli accessi 277
4.3.6 Il *Name Service Switch* 281
4.3.7 I *Pluggable Authentication Modules* 284
4.4 Amministrazione sistemistica di una base di dati 292
4.4.1 Alcuni concetti base dei database relazionali 292
4.4.2 Configurazione e gestione di base di PostgreSQL 294
4.4.3 Configurazione e gestione di base di MySQL 298
4.4.4 Nozioni elementari di SQL 301

5 Amministrazione straordinaria del sistema 307
5.1 La gestione dei dischi e dei filesystem 307
5.1.1 Alcune nozioni generali 307
5.1.2 Il partizionamento dei dischi 308
5.1.3 La gestione del contenuto dei dischi 317
5.1.4 La creazione di un filesystem 327
5.1.5 Controllo e riparazione di un filesystem 335
5.1.6 Il sistema dell'*automounter* 342
5.1.7 La gestione della *swap* e dei CDROM 344
5.2 La gestione di kernel e moduli 348
5.2.1 Le versioni del kernel 348
5.2.2 Sorgenti e *patch* 350
5.2.3 La ricompilazione del kernel 353
5.2.4 Installazione manuale del kernel e del *ramdisk* iniziale 365
5.2.5 La gestione dei moduli 370
5.3 La gestione dell'avvio del sistema 378
5.3.1 L'architettura dell'avvio del sistema 378
5.3.2 Il *bootloader* SYSLINUX 381
5.3.3 I *bootloader* GRUB e GRUB2 383
5.3.4 Lancio del kernel, *ramdisk* ed avvio del sistema 391
5.3.5 Il sistema di avvio alla *System V* 393
5.3.6 Sistemi di avvio alternativi e `systemd` 400
5.3.7 Gestione generica di servizi, riavvio, spegnimento 411
5.4 La gestione di interfacce e periferiche 413
5.4.1 Gestione delle interfacce di espansione 413
5.4.2 Gestione delle interfacce SCSI 419
5.4.3 Gestione delle interfacce seriali 423
5.4.4 Gestione delle interfacce USB 426
5.4.5 La gestione dei dispositivi con *udev* e *hotplug* 429

6 Amministrazione avanzata del sistema **437**
6.1 L'utilizzo del RAID 437
6.1.1 Introduzione 437
6.1.2 Il RAID su Linux 441
6.1.3 La gestione del RAID software 442
6.2 Il sistema del *Logical Volume Manager* 452
6.2.1 Introduzione 453
6.2.2 La gestione dei *volumi fisici* 455
6.2.3 La gestione dei *gruppi di volumi* 457
6.2.4 La gestione dei *volumi logici* 460
6.3 Gestione avanzata di dischi e filesystem 462
6.3.1 La gestione dei parametri dei dischi 462
6.3.2 Il ridimensionamento di filesystem e partizioni 468
6.3.3 Dischi e filesystem cifrati 472
6.4 Le quote disco 476
6.4.1 Visione generale 477
6.4.2 Configurazione del sistema delle quote 477
6.4.3 Gestione delle quote di utenti e gruppi 479

7 Amministrazione di base delle reti **483**
7.1 Un'introduzione ai concetti fondamentali delle reti. 483
7.1.1 L'estensione 483
7.1.2 La topologia 484
7.1.3 I protocolli 486
7.2 Il TCP/IP. 490
7.2.1 Introduzione. 490
7.2.2 Gli indirizzi IP (IPv4 e IPv6) 494
7.2.3 L'instradamento o *routing* 499
7.2.4 I servizi e le porte. 501
7.3 La configurazione di base 503
7.3.1 L'assegnazione degli indirizzi ed il comando `ifconfig` 504
7.3.2 L'impostazione dell'instradamento ed il comando `route` 508
7.3.3 La configurazione della rete all'avvio del sistema. 513
7.3.4 Comandi di configurazione avanzata 519
7.4 Il sistema della risoluzione dei nomi 526
7.4.1 Introduzione al *resolver* 526
7.4.2 I file di configurazione del *resolver* 528
7.4.3 La gestione locale dei nomi 531
7.4.4 La gestione degli altri nomi di rete 532
7.5 Altre tipologie di connessioni di rete 534
7.5.1 Cenni sul protocollo PPP 534
7.5.2 Il demone `pppd` 535
7.5.3 I meccanismi di autenticazione 539
7.5.4 La connettività *Wi-Fi* 540
7.5.5 I *wireless tool* di Linux 541

7.6 I comandi diagnostici . . . 544
7.6.1 Il comando `ping` . . . 544
7.6.2 I comandi `traceroute`, `tracepath` e `mtr` . . . 545
7.6.3 Il comando `netstat` . . . 549
7.6.4 Il protocollo ARP ed il comando `arp` . . . 551
7.6.5 I servizi RPC . . . 553
7.7 I client dei servizi di base . . . 555
7.7.1 I comandi `telnet` e `netcat` . . . 555
7.7.2 Il comando `ftp` . . . 558
7.7.3 I comandi `finger` e `whois` . . . 559

8 La gestione dei servizi di base 563
8.1 I programmi di ausilio alla gestione dei servizi di base . . . 563
8.1.1 I servizi elementari e i super-demoni . . . 563
8.1.2 Il super-demone `inetd` . . . 564
8.1.3 Il super-demone `xinetd` . . . 567
8.1.4 Il servizio NTP . . . 572
8.1.5 I TCP wrappers . . . 574
8.2 L'assegnazione dinamica degli indirizzi IP . . . 577
8.2.1 I protocolli RARP, BOOTP e DHCP . . . 577
8.2.2 Uso del servizio DHCP dal lato client . . . 580
8.2.3 La configurazione di un server DHCP . . . 583
8.3 Amministrazione remota con SSH . . . 585
8.3.1 La configurazione del server SSH . . . 586
8.3.2 L'utilizzo di SSH come client . . . 589
8.3.3 La autenticazione a chiavi . . . 594
8.3.4 Le funzionalità di reinoltro del traffico . . . 598
8.4 La condivisione dei file sulla rete . . . 600
8.4.1 Il protocollo ed il server NFS . . . 600
8.4.2 NFS sul lato client . . . 603
8.4.3 La configurazione di base di Samba . . . 605
8.5 La gestione elementare della posta elettronica . . . 613
8.5.1 La struttura del servizio della posta elettronica . . . 613
8.5.2 La posta elettronica sulla riga di comando . . . 616
8.5.3 Le funzionalità di compatibilità con *Sendmail* . . . 617

9 Il servizio DNS 621
9.1 Il funzionamento del servizio DNS . . . 621
9.1.1 Introduzione . . . 621
9.1.2 I *Resorce Record* e le *zone di autorità* . . . 624
9.1.3 I comandi `host` e `dig` . . . 625
9.2 La gestione di un server DNS . . . 628
9.2.1 Il server *BIND* . . . 628
9.2.2 Il file `named.conf` . . . 629
9.2.3 La configurazione base di *BIND* . . . 631

9.2.4 La configurazione di un dominio locale. 634
9.3 Configurazioni avanzate 639
9.3.1 La delegazione di una zona 639
9.3.2 La gestione di un secondario 640

A Indice degli argomenti per LPI 643
A.1 Argomenti Certificazione LPIC-1 643
A.1.1 Argomenti Esame LPI 101 643
A.1.2 Argomenti Esame LPI 102 644
A.2 Argomenti Certificazione LPIC-2 644
A.2.1 Argomenti Esame LPI 201 645
A.2.2 Argomenti Esame LPI 202 645
A.3 Percorsi di studio per la certificazione LPI 646
A.3.1 Percorso di studio per l'esame LPI 101 646
A.3.2 Percorso di studio per l'esame LPI 102 647

B Ringraziamenti 649

C GNU Free Documentation License 651
C.1 Applicability and Definitions 652
C.2 Verbatim Copying 653
C.3 Copying in Quantity 653
C.4 Modifications 654
C.5 Combining Documents 655
C.6 Collections of Documents 656
C.7 Aggregation With Independent Works 656
C.8 Translation 656
C.9 Termination 657
C.10 Future Revisions of This License 657
C.11 Relicensing 657
C.12 Addendum: How to use this License for your documents 658

Capitolo 1

L'architettura di un sistema GNU/Linux

1.1 L'architettura del sistema.

Prima di addentrarci nei dettagli dell'amministrazione di un sistema GNU/Linux, conviene fornire un quadro generale per introdurre i vari concetti su cui si fonda l'architettura di questo sistema, che nasce da quella, consolidatasi in 30 anni di impiego, dei sistemi di tipo Unix.

Il fatto che questa architettura abbia una certa età fa sì che spesso i detrattori di GNU/Linux ne denuncino la presunta mancanza di *innovatività*, ma anche le case hanno da secoli le stesse basi architetturali (porte, muri e tetti), ma non per questo non esiste innovazione. Il vantaggio dell'architettura di Unix infatti è quello di aver fornito una solida base per la costruzione di sistemi affidabili ed efficienti, consolidata e corretta in decenni di utilizzo, tanto che, per citare Henry Spencer: "*those who don't understand UNIX are doomed to reinvent it, poorly*".

1.1.1 L'architettura di base.

Contrariamente ad altri sistemi operativi, GNU/Linux nasce, come tutti gli Unix, come sistema multitasking e multiutente. Questo significa che GNU/Linux ha un'architettura di sistema che è stata pensata fin dall'inizio per l'uso contemporaneo da parte di più utenti. Questo comporta conseguenze non del tutto intuitive nel caso in cui, come oggi accade sempre più spesso, esso venga usato come stazione di lavoro da un utente singolo.

Il concetto base dell'architettura di ogni sistema Unix come GNU/Linux è quello di una rigida separazione fra il *kernel* (il *nucleo* del sistema), cui si demanda la gestione delle risorse hardware (come la CPU, la memoria, le periferiche) ed i *processi*, (le unità di esecuzione dei programmi), che nel caso vanno dai comandi base di sistema, agli applicativi, alle interfacce per l'interazione con gli utenti.

Lo scopo del kernel infatti è solo quello di essere in grado di eseguire contemporaneamente molti processi in maniera efficiente, garantendo una corretta distribuzione fra gli stessi della memoria e del tempo di CPU, e quello di fornire le adeguate interfacce software per l'accesso

alle periferiche della macchina e le infrastrutture di base necessarie per costruire i servizi. Tutto il resto, dall'autenticazione all'interfaccia utente, viene realizzato usando processi che eseguono gli opportuni programmi.

Questo si traduce in una delle caratteristiche essenziali su cui si basa l'architettura dei sistemi Unix: la distinzione fra il cosiddetto *user space*, che è l'ambiente a disposizione degli utenti, in cui vengono eseguiti i processi, e il *kernel space*, che è l'ambiente in cui viene eseguito il kernel. I due ambienti comunicano attraverso un insieme di interfacce ben definite e standardizzate; secondo una struttura come quella mostrata in fig. 1.1.

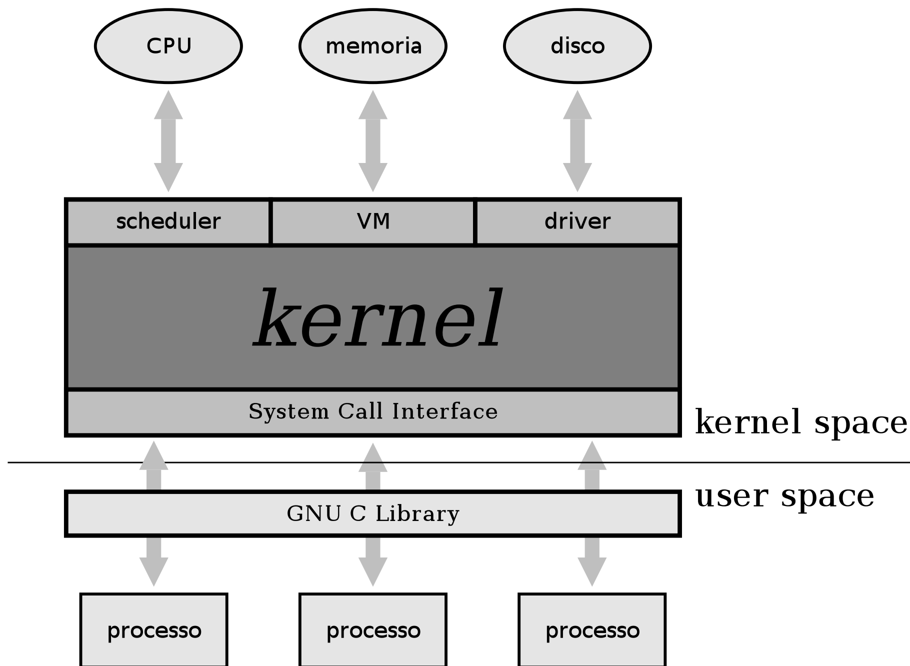


Figura 1.1: Schema della struttura del sistema operativo GNU/Linux.

Questa architettura comporta che solo il kernel viene eseguito in modalità privilegiata, ed è l'unico a poter accedere direttamente alle risorse dell'hardware. I normali programmi invece vengono eseguiti in modalità protetta, in un ambiente virtuale, l'*user space*, in cui essi vedono se stessi come se avessero piena disponibilità della CPU e della memoria, ma in cui possono accedere alle periferiche e alle altre funzionalità messe a disposizione del kernel solo attraverso una serie di *funzioni di sistema* standardizzate, dette *system call*. Lo standard in questo caso si chiama POSIX.1; ma oltre a quelle dello standard Linux supporta ulteriori *system call*, relative a sue estensioni specifiche.

Le *system call* e tutta una serie di ulteriori funzioni di base sono tradizionalmente raccolte in un'unica libreria che pertanto diventa essenziale per il funzionamento di qualunque programma (tratteremo le librerie, e come vengono gestite nel sistema, in sez. 3.1.2). Nel caso di Linux questa libreria, illustrata nello schema in fig. 1.1, è la *GNU C library* (chiamata in breve *glibc*), che costituisce una delle parti fondamentali del sistema.

In sostanza quello che succede è che da un certo punto di vista l'unico "*vero*" programma che viene eseguito è il kernel, che si incarica di costruire questo ambiente virtuale in cui far girare gli altri programmi. Una parte del kernel, quella indicata in fig. 1.1 come scheduler, è deputata della gestione del tempo di processore, e provvederà a decidere volta per volta qual è il processo che deve essere eseguito in un determinato momento (realizzando così il multitasking).

Una seconda parte, quella indicata in fig. 1.1 come VM (sigla che sta per *Virtual Memory*), si occupa invece di gestire l'uso della memoria disponibile. La *memoria virtuale* è uno dei sottosistemi più importanti del kernel, oggetto di continui miglioramenti in quanto critico per tutte le prestazioni del sistema, perché è quella che fa in modo che ogni processo veda uno spazio di indirizzi proprio che poi viene rimappato nella memoria fisica effettivamente presente,[1] così che sia impossibile che un processo possa accedere alla memoria di un altro processo. La *memoria virtuale* si incarica anche di gestire, in caso di esaurimento della RAM, l'eventuale spostamento delle pagine di memoria meno usate su uno opportuno spazio disco (lo *swap*, che tratteremo in sez. 5.1.7) evitando di fermare l'esecuzione di un processo per una temporanea mancanza di memoria. Torneremo brevemente sull'argomento in sez. 1.3.2.

Infine c'è un'ultima parte del kernel, indicata in fig. 1.1 con l'indicazione generica driver anche se in realtà non è un unico sottosistema come le precedenti ma, come vedremo in sez. 5.2.5, un insieme di "*moduli*" che consentono la gestione delle varie periferiche, fornendo un accesso alle stesse per conto dei programmi. Questa parte, come vedremo meglio in sez. 1.2, permette di definire una interfaccia di accesso generica per i dispositivi, cui spesso si fa riferimento dicendo che in un sistema unix-like "*tutto è un file*".

La conseguenza più importante di questa separazione fra *user space* e *kernel space* è che in questo modo non è possibile che un singolo programma possa disturbare l'azione di un altro programma o del kernel stesso, e questo è il principale motivo della stabilità di un sistema Unix nei confronti di altri sistemi in cui i processi non hanno di questi limiti, o vengono, per vari motivi, eseguiti all'interno del kernel.

1.1.2 Il funzionamento del sistema

Per illustrare meglio la distinzione fra *kernel space* e *user space* prendiamo brevemente in esame la procedura di avvio del sistema, su cui torneremo in dettaglio in sez. 5.3. All'accensione del computer viene eseguito il programma del BIOS; questo dopo aver fatto i suoi controlli interni esegue la procedura di avvio del sistema. Nei PC tutto ciò viene effettuato caricando dal dispositivo indicato nelle impostazioni del BIOS un apposito programma, il *bootloader*,[2] che a sua volta recupera (in genere dal disco) una immagine del kernel che viene caricata in memoria ed eseguita.

Una volta che il controllo è passato al kernel questo, terminata la fase di inizializzazione in cui ad esempio si esegue una scansione delle periferiche disponibili, e si leggono le tabelle delle partizioni dei vari dischi (vedi sez. 5.1.2) si incaricherà di montare (vedi sez. 1.2.2) il filesystem su cui è situata la *directory radice* (vedi sez. 1.2.3), e farà partire il primo processo.

[1] questo viene in genere realizzato con l'ausilio delle MMU (*Memory Management Unit*) dei microprocessori; una descrizione più dettagliata del funzionamento della *memoria virtuale* può essere trovata nella sezione 2.2 di [GaPiL].

[2] questo è un programma speciale, il cui solo compito è quello di far partire un sistema operativo, in genere ogni sistema ha il suo, nel caso di Linux per l'architettura PC tratteremo i più comuni in sez. 5.3.

Tradizionalmente questo processo si chiama `init`, ed è il programma di inizializzazione che a sua volta si cura di far partire tutti gli altri processi che permettono di usare il sistema.

Fra questi processi ci saranno ad esempio quelli che forniscono i vari servizi di rete, quelli che eseguono vari compiti di amministrazione, così come quelli che si occupano di chiedere nome e password dell'utente che si vuole collegare,[3] e che una volta completato l'ingresso nel sistema (procedura che viene chiamata *login*) lanciano altri programmi per mettere a disposizione dell'utente l'interfaccia da cui inviare i comandi, che potrebbe essere sia una shell a riga di comando (argomento che tratteremo in dettaglio in cap. 2) che una delle tante interfacce grafiche disponibili (argomento che riprenderemo in sez. 3.3).

È da rimarcare poi come anche tutti i programmi che un utente di un sistema GNU/Linux può utilizzare una volta che si è collegato non hanno niente di diverso da quelli appena citati. Tutti i programmi funzionano allo stesso modo: vengono eseguiti dal kernel come processi ed eseguono le loro operazioni attraverso le opportune *system call* che esso mette a disposizione. Da questo punto di vista eseguire sulla shell un programma per vedere la lista dei file non ha niente di diverso dall'eseguire in ambiente grafico un programma di scrittura o un programma di foto-ritocco, o dal lanciare un server web che viene eseguito anche quando nessuno è collegato al sistema.

Questo significa ad esempio che il kernel di per sé non dispone di primitive per tutta una serie di operazioni, come la copia di un file,[4] che altri sistemi operativi (come Windows) hanno al loro interno: tutte le operazioni di normale amministrazione di un sistema GNU/Linux sono sempre realizzate tramite degli opportuni programmi.

Tutto ciò ci dice anche che benché esso costituisca il cuore del sistema, il kernel da solo sarebbe assolutamente inutile, così come sarebbe inutile da solo il motore di una automobile, senza avere le ruote, lo sterzo, la carrozzeria, e tutto il resto. Per avere un sistema funzionante dal punto di vista di un utente normale infatti occorre avere, oltre al kernel, anche tutti i programmi che gli permettano di eseguire le varie operazioni con i dischi, i file, le periferiche.

Per questo al kernel vengono sempre uniti degli opportuni programmi di gestione per il sistema e tutta una serie di programmi applicativi, ed è l'insieme di questi e del kernel che costituisce un sistema funzionante. Di solito i rivenditori, o anche gruppi di volontari, come nel caso di Debian, si preoccupano di raccogliere in forma coerente i programmi necessari, per andare a costruire quella che viene chiamata una *distribuzione*. Sono in genere queste distribuzioni (come Debian, RedHat, Slackware, SuSE, Ubuntu[5]), quelle che si trovano sui CD con i quali si installa quello che, con una semplificazione molto brutale, viene chiamato solo "Linux".

Il gruppo principale di questi programmi, e le librerie di base che essi e tutti gli altri programmi usano, derivano dal progetto GNU della Free Software Foundation: è su di essi che ogni altro programma è basato, ed è per questo che è più corretto riferirsi all'intero sistema come a GNU/Linux, dato che Linux indica solo una parte, il kernel, che benché fondamentale non costituisce da sola un sistema operativo.

Si tenga presente infine che anche se il kernel tratta tutti i programmi allo stesso modo, non tutti hanno la stessa importanza. Nella precedente descrizione dell'avvio del sistema abbiamo

[3] in realtà se la cosa è fatta da *console* i programmi sono due, il primo chiede l'utente e poi chiama il secondo che chiede la password, torneremo su questo in sez. 4.3.5 e sez. 5.3.5.

[4] questa infatti viene eseguita usando semplicemente le funzioni che permettono di leggere e scrivere il contenuto di un file, leggendo l'originale e scrivendo sulla copia.

[5] in rigoroso ordine alfabetico!

accennato ad un programma particolare, `init`, che ha un ruolo privilegiato in quanto è quello che si occupa dell'inizializzazione del sistema. La sua peculiarità è quella di essere lanciato per primo direttamente dal kernel e dover gestire il lancio di tutti gli altri programmi, per cui non può essere mai terminato. Ma anche `init` alla fine non è che un programma che usa le *system call* e viene eseguito dal kernel, come un qualunque altro programma, ed infatti esistono alternative rispetto alla implementazione classica, come `systemd` o `upstart` su cui torneremo in sez. 5.3.

Questo ci porta ad un'altra caratteristica fondamentale dell'architettura dei sistemi unix-like (ci torneremo in dettaglio in sez. 1.3) che è quella per cui qualunque processo può a sua volta lanciarne di nuovi, per cui si dice che il primo processo è il *padre* degli altri, che a loro volta sono chiamati *figli*. È questo che permette l'avvio del sistema eseguendo un unico programma di inizializzazione (come `init` o alternative), dato che quest'ultimo potrà poi lanciare altri programmi, che a loro volta ne potranno lanciare degli altri ancora, fino a fornire tutte le funzionalità richieste per il funzionamento del sistema.

Benché sia possibile per usi particolari (ad esempio in sistemi *embedded*[6] che devono svolgere un solo compito) far partire un programma applicativo al posto di `init`, e anche se, come vedremo in sez. 5.3, in casi di emergenza si può lanciare al suo posto una shell, in pratica tutti i sistemi Unix usano un programma dedicato per gestire l'avvio del sistema, ed è a seconda degli ulteriori programmi che questo mette in esecuzione che alla fine della procedura di avvio ci si troverà davanti ad un terminale a caratteri o ad una interfaccia grafica, e si avrà, a seconda di quanto deciso (ed installato) dall'amministratore, un server di posta, un server web, una workstation, ecc.

1.1.3 Alcune caratteristiche specifiche di Linux

Benché Linux stia diventando il più diffuso, esistono parecchi altri kernel unix-like, sia liberi che proprietari, nati nella tumultuosa e complessa evoluzione che dallo Unix originario della AT/T ha portato alla nascita di una miriade di sistemi derivati (BSD, Solaris, AIX, HP-UX, Digital Unix, IRIX, solo per citare i più noti) che si innestano tutti in due rami principali, quelli derivati dal sistema sviluppato dalla AT/T, detto SysV (da *System V*, l'ultima versione ufficiale) e quelli derivati dal codice sviluppato all'università di Berkeley, detto BSD (da *Berkeley Software Distribution*). La prima caratteristica distintiva di Linux è che esso è stato riscritto da zero, per cui non è classificabile in nessuno di questi due rami e prende invece, a seconda dei casi, quelle che sono state ritenute le migliori caratteristiche di ciascuno di essi.

Un'altra delle caratteristiche peculiari di Linux rispetto agli altri kernel unix-like è quella di essere *modulare*; Linux cioè può essere esteso (torneremo su questo in sez. 5.2.5) inserendo a sistema attivo degli ulteriori "pezzi", i *moduli*, che permettono di ampliare le capacità del sistema (ad esempio fargli riconoscere una nuova periferica). Questi possono poi essere tolti dal sistema quando non sono più necessari: un caso tipico può essere quello del modulo che permette di vedere il CDROM, viene caricato solo quando c'è necessità di leggere un CD e può essere rimosso una volta che questo non sia più in uso.

In realtà è sempre possibile costruire un kernel Linux comprensivo di tutti i moduli che servono, ottenendo quello che viene chiamato un kernel *monolitico* (come sono i kernel tradizionale degli altri Unix). Questo permette di evitare il ritardo nel caricamento dei moduli al momento

[6] si chiamano così i sistemi destinati all'esecuzione di compiti specifici, come quelli dei telefonini, dei videoregistratori, ecc.

della richiesta, ma comporta un maggiore consumo di memoria, dovendo tenere dentro il kernel anche codice non utilizzato, ed una flessibilità nettamente inferiore, in quanto si perde la capacità di poter specificare eventuali opzioni al momento del caricamento, costringendo al riavvio in caso di necessità di cambiamenti.

Per contro in certi casi l'uso dei moduli può degradare leggermente le prestazioni, quasi sempre in maniera non avvertibile, e può dar luogo a conflitti inaspettati che con un kernel monolitico avrebbero bloccato il sistema all'avvio, ma questi problemi oggi sono sempre più rari. In ogni caso non è possibile utilizzare i moduli nel caso in cui le funzionalità da essi fornite siano necessarie ad avviare il sistema.

Una terza peculiarità di Linux è quella del *Virtual File System* (o VFS). Un concetto generale presente in tutti i sistemi Unix (e non solo) è che lo spazio su disco su cui vengono tenuti i file di dati è organizzato in quello che viene chiamato un *filesystem* (tratteremo l'amministrazione dei *filesystem* in sez. 5.1). Lo spazio disco grezzo è normalmente suddiviso in settori contigui di dimensione fissa,[7] ma all'interno del sistema questo viene organizzato in maniera tale da permettere il rapido reperimento delle informazioni memorizzate su questi settori, anche quando queste sono sparse qua e là sul disco; si ha così quello che l'utente vede come un singolo file.

Quello che contraddistingue Linux è che l'interfaccia per la lettura del contenuto di un filesystem è stata completamente virtualizzata, per cui inserendo gli opportuni moduli nel sistema diventa possibile accedere con la stessa interfaccia e, salvo limitazioni della realizzazione, in maniera completamente trasparente all'utente ai più svariati tipi di filesystem, a partire da quelli usati da Windows e dal DOS, dal MacOS, e da tutte le altre versioni di Unix.

Dato che essa gioca un ruolo centrale nel sistema, torneremo in dettaglio sull'interfaccia dei file, e di come possa essere usata anche per altro che i file di dati, in sez. 1.2. Quello che è importante tenere presente da subito è che la disponibilità di una astrazione delle operazioni sui file rende Linux estremamente flessibile, dato che attraverso di essa è stato in grado, fin dalle prime versioni, di supportare con relativa facilità ed in maniera nativa una quantità di filesystem superiore a quella di qualunque altro sistema operativo.

1.2 L'architettura dei file

Un aspetto fondamentale dell'architettura di GNU/Linux è quello della gestione dei file. Esso deriva direttamente da uno dei criteri base della progettazione di tutti i sistemi Unix, quello espresso dalla frase "*everything is a file*" (cioè *tutto è un file*), per cui l'accesso ai file e alle periferiche è gestito attraverso una interfaccia identica.

Inoltre, essendo in presenza di un sistema multiutente e multitasking, il kernel deve anche essere in grado di gestire l'accesso contemporaneo allo stesso file da parte di più processi, e questo viene fatto usando un disegno specifico nella struttura delle interfacce di accesso, che è uno dei punti di maggior forza dell'architettura di un sistema Unix.

1.2.1 Il *Virtual File System* e le caratteristiche dei file.

Come accennato in sez. 1.1.3 i file sono organizzati sui dischi all'interno di filesystem. Perché i file diventino accessibili al sistema un filesystem deve essere *montato* in modo che essi siano resi

[7] nel senso che le interfacce hardware per i dischi consentono l'accesso diretto al contenuto di questi settori.

visibili all'interno di una certa directory. Questa è una operazione privilegiata che normalmente può fare solo l'amministratore (ne tratteremo i dettagli più avanti, in sez. 5.1.3), che provvede a rendere disponibili le interfacce, in genere attraverso il caricamento di opportuni moduli del kernel, che permettono l'accesso ai file contenuti nel filesystem.

Come esempio consideriamo il caso in cui si voglia leggere il contenuto di un CD. Il kernel dovrà poter disporre sia delle interfacce per poter parlare al dispositivo fisico (ad esempio il supporto SCSI, se il CDROM è SCSI), che di quelle per la lettura dal dispositivo specifico (il modulo che si interfaccia ai CDROM, che è lo stesso che questi siano su SCSI, IDE o USB), sia di quelle che permettono di interpretare il filesystem ISO9660 (che è quello che di solito viene usato per i dati registrati su un CDROM) per estrarne il contenuto dei file.

Come accennato nell'introduzione a questa sezione, uno dei criteri fondamentali dell'architettura di un sistema Unix è quello per cui *tutto è un file* e per cui si può accedere a tutte le periferiche, con la sola eccezione delle interfacce ai dispositivi di rete, che non rientrano bene nell'astrazione e sono gestite in maniera diversa, con una interfaccia identica a quella con cui si accede al contenuto dei file. Questo comporta però una serie di differenze nella gestione dei file rispetto ad altri sistemi.

Anzitutto in un sistema Unix tutti i file di dati sono uguali: non esiste cioè la differenza fra file di testo o binari che c'è in Windows, né fra file sequenziali e ad accesso diretto che c'era nel VMS. Inoltre le estensioni sono solo convenzioni, e non significano nulla per il kernel, che legge tutti i file di dati alla stessa maniera, indipendentemente dal nome e dal contenuto.

In realtà il sistema prevede tipi diversi di file, ma in un altro senso; ad esempio il sistema può accedere alle periferiche, attraverso dei file speciali detti *device file* o *file di dispositivo*. Così si può suonare una canzone scrivendo su `/dev/dsp`,[8] leggere l'output di una seriale direttamente da `/dev/ttyS0`, leggere direttamente dai settori fisici del disco rigido accedendo a `/dev/hda`, o fare animazioni sullo schermo scrivendo su `/dev/fb0` (questo è molto più difficile da fare a mano). Un elenco dei vari oggetti visti come file dal kernel è riportato in tab. 1.1; ognuno di questi fornisce una funzionalità specifica, sempre descritta nella medesima tabella.

Altri due tipi di file speciali riportati in tab. 1.1 sono le *fifo* ed i *socket*, che altro non sono che dei canali di comunicazione messi a disposizione dal kernel ai processi, perché questi possano parlarsi fra di loro. Dato che, come accennato in sez 1.1, i processi sono completamente separati fra loro, può essere soltanto il kernel a fornire le funzionalità che permettano la comunicazione.

Questi file speciali sono due modalità diverse per realizzare una comunicazione fra processi diversi; aprendo una *fifo* un processo può scrivervi sopra ed un altro processo che ha aperto la *fifo* leggerà dall'altro capo quanto il primo ha scritto,[9] niente verrà salvato su disco, ma passerà tutto attraverso il kernel che consente questa comunicazione come attraverso un *tubo* (da questo deriva il nome *pipe*). I *socket* fanno la stessa cosa ma consentono una comunicazione bidirezionale, in cui il secondo processo può scrivere indietro, ed il primo leggere, quando invece per le *fifo* il flusso dei dati è unidirezionale.

[8] questo avveniva con il vecchio supporto audio, con ALSA (la nuova infrastruttura del kernel per i dispositivi audio) i nomi dei dispositivi sono diversi ed in genere oggi la gestione dell'audio viene comunque delegata ad un programma che fa da interfaccia fra i dispositivi, che possono essere i più vari, e gli altri programmi che vogliono usare il suono.

[9] l'uso delle *fifo* su disco è in realtà abbastanza complesso e benché quasi sempre coinvolga due soli processi (uno che scrive e uno che legge) si possono avere situazioni più complicate; la trattazione dell'argomento va oltre gli scopi di questo testo, chi fosse interessato può trovare delle spiegazioni dettagliate sull'argomento nella sezione 9.1 di [GaPiL].

Tipo di file			Descrizione
regular file	*file regolare*	`-`	un file che contiene dei dati (ciò che si intende normalmente per file).
directory	*cartella o direttorio*	`d`	un file che contiene una lista di nomi associati a degli *inode*.
symbolic link	*collegamento simbolico*	`l`	un file che contiene un riferimento ad un altro file o directory.
char device	*dispositivo a caratteri*	`c`	un file che identifica una periferica ad accesso a caratteri.
block device	*dispositivo a blocchi*	`b`	un file che identifica una periferica ad accesso a blocchi.
fifo o *pipe*	"*coda*" o "*tubo*"	`p`	un file speciale che identifica una linea di comunicazione unidirezionale.
socket	"*presa*"	`s`	un file speciale che identifica una linea di comunicazione bidirezionale.

Tabella 1.1: I vari tipi di file riconosciuti da Linux

La possibilità di avere tutti questi tipi di file speciali è dovuta al fatto che, come accennavamo in sez. 1.1.3, in Linux l'accesso ai file viene eseguito attraverso una interfaccia unificata, il *Virtual File System*, che definisce una serie di operazioni generiche che sono le stesse per tutti gli oggetti che essa gestisce; è questo che implementa la filosofia progettuale del *tutto è un file*.

Le principali operazioni sono riportate in tab. 1.2; ogni oggetto del sistema visto attraverso il *Virtual File System* definisce la sua versione di queste operazioni. Come si può notare sono definite sia operazioni generiche come la lettura e la scrittura, che più specialistiche, come lo spostamento all'interno di un file.[10] Quando si utilizzano le *system call* per accedere ad un file sarà compito del kernel chiamare l'operazione relativa ad esso associata (che sarà ovviamente diversa a seconda del tipo di file), o riportare un errore quando quest'ultima non sia definita (ad esempio sul file di dispositivo associato alla seriale non si potrà mai utilizzare l'operazione di spostamento *`llseek`*).

Funzione	Operazione
`open`	apre il file.
`read`	legge dal file.
`write`	scrive sul file.
`llseek`	si sposta all'interno del file.
`ioctl`	accede alle operazioni di controllo.
`readdir`	legge il contenuto di una directory.

Tabella 1.2: Principali operazioni sui file definite nel VFS.

Il *Virtual File System* è anche il meccanismo che permette al kernel di gestire tanti filesystem diversi; quando uno di questi viene montato è compito del kernel utilizzare per le varie *system call* le opportune operazioni in grado di accedere al contenuto di quel particolare filesystem; questa è la ragione principale della grande flessibilità di Linux nel supportare i filesystem più diversi, basta definire queste operazioni per un filesystem per poterne permettere l'accesso da parte delle varie *system call* secondo la stessa interfaccia.

[10] ce ne sono altre ancora più complesse, che non sono state riportate in tab. 1.2, per una trattazione dettagliata dell'argomento si consulti sez. 4.1.1 di [GaPiL].

Uno dei programmi fondamentali per la gestione dei file attraverso l'interfaccia a riga di comando è `ls` (il cui nome, non proprio intuitivo, deriva da ***list** file*) che mostra, se invocato senza nessun argomento, l'elenco dei file nella directory corrente. Usando l'opzione `-l` è invece possibile ottenere una lista estesa, in cui compaiono varie proprietà dei file. Ad esempio, avendo creato una directory con i vari tipi di file citati in tab. 1.1,[11] si potrà andare in detta directory ed ottenere:

```
piccardi@oppish:~/filetypes$ ls -l
total 1
brw-r--r--    1 root     root       1,   2 Jul  8 14:48 block
crw-r--r--    1 root     root       1,   2 Jul  8 14:48 char
drwxr-xr-x    2 piccardi piccardi      48 Jul  8 14:24 dir
prw-r--r--    1 piccardi piccardi       0 Jul  8 14:24 fifo
-rw-r--r--    1 piccardi piccardi       0 Jul  8 14:24 file
lrwxrwxrwx    1 piccardi piccardi       4 Jul  8 14:25 link -> file
```

e si noti come la prima lettera in ciascuna riga indichi il tipo di file, anche questo secondo la notazione riportata nella terza colonna di tab. 1.1.

Il comando `ls` è dotato di innumerevoli opzioni, che gli consentono di visualizzare le varie caratteristiche dei file, delle quali il tipo è solo una. Altre caratteristiche sono: i tempi di ultimo accesso, modifica e cambiamento (su cui torneremo fra poco), le informazioni relative a permessi di accesso e proprietari del file (che vedremo in dettaglio in sez. 1.4.2), la dimensione, il numero di *hard link* (che vedremo in sez. 1.2.2).

Il comando prende come argomento una lista di file o directory; senza opzioni viene mostrato solo il nome del file (se esiste) o il contenuto della directory specificata. Le opzioni sono moltissime, e le principali sono riportate in tab. 1.3, l'elenco completo è riportato nella pagina di manuale accessibile con il comando `man ls`.

Opzione	**Significato**
`-l`	scrive la lista in formato esteso.
`-a`	mostra i file *invisibili*.
`-i`	scrive il numero di *inode* (vedi sez. 1.2.2).
`-R`	esegue la lista ricorsivamente per tutte le sottodirectory.
`-c`	usa il tempo di ultimo cambiamento del file.
`-u`	usa il tempo di ultimo accesso al file.
`-d`	mostra solo il nome e non il contenuto quando riferito ad una directory, e non segue i link simbolici.

Tabella 1.3: Principali opzioni del comando `ls`.

Una convenzione vuole che i file il cui nome inizia con un punto ("`.`") non vengano riportati nell'output di `ls`, a meno di non richiederlo esplicitamente con l'uso dell'opzione `-a`; per questo tali file sono detti *invisibili*. Si tenga presente comunque che questa *non* è una proprietà dei file e non ha nulla a che fare con le modalità con cui il kernel li tratta (che sono sempre le stesse), ma solo una convenzione usata e rispettata dai vari programmi in *user space*.

L'opzione `-l` permette di mostrare una lista in formato esteso in cui vengono riportate molte informazioni concernenti il file. Abbiamo visto in precedenza un esempio di questa lista, e come

[11] con l'eccezione dei *socket*, questi ultimi infatti non sono di norma utilizzati dai comandi di shell, ma vengono creati direttamente dai programmi che li usano (il caso più comune è X Window), non esiste pertanto un comando di uso comune che permetta di crearne uno in maniera esplicita.

il primo carattere della prima colonna indichi il tipo di file, mentre il resto della colonna indica i *permessi* del file, secondo una notazione su cui torneremo in sez. 1.4.2. Il secondo campo indica il numero di *hard link* al file (su questo torneremo in sez. 1.2.2), mentre il terzo ed il quarto campo indicano rispettivamente utente e gruppo proprietari del file (anche questo sarà trattato in sez. 1.4.2). Il quinto campo indica la dimensione, usualmente riportata in byte. Il sesto campo è il tempo di *ultima modifica* del file e l'ultimo campo il nome del file.

In un sistema unix-like i tempi associati ai file (mantenuti automaticamente dal kernel quando opera su di essi) sono tre ed hanno un significato diverso rispetto a quanto si trova in altri sistemi operativi.[12] Il tempo mostrato di default da `ls` è il cosiddetto *tempo di ultima modifica* (o *modification time*) che corrisponde all'ultima volta che è stato modificato il contenuto di un file.

Si badi bene che questo tempo riguarda solo il *contenuto* del file, se invece si operano delle modifiche sulle proprietà del file (ad esempio si cambiano i permessi) varia quello che viene chiamato *tempo di ultimo cambiamento* (il *change time* o *status change time*) che viene visualizzato da `ls` con l'opzione `-c`. Infine tutte le volte che si accede al contenuto del file viene cambiato[13] il *tempo di ultimo accesso* (o *access time*) che può essere visualizzato da `ls` con l'opzione `-u`.

Si noti infine come fra i precedenti tempi non ci sia un tempo di creazione del file, che in un sistema unix-like *non* esiste come informazione a parte. Quando un file viene creato infatti vengono impostati i tre tempi precedenti al tempo corrente, ma questa informazione verrà sovrascritta nelle operazioni successive; fintanto che non si modificano i contenuti comunque si potrà considerare il tempo di ultima modifica come coincidente con il tempo di creazione.

Per operare sui tempi associati ad un file esiste un apposito comando, `touch`. In genere questo comando viene usato in quasi tutti gli esempi per creare un file vuoto, ma in realtà il suo scopo non è questo (dato che lo stesso compito si potrebbe eseguire in molti altri modi) quanto piuttosto, come dice il nome, quello di *toccare* un file, anche già esistente, cambiandone i tempi.

Il comando prende come argomenti uno o più nomi di file. Se uno di questi non esiste il risultato del comando è quello di crearlo vuoto, ma se invece esiste l'effetto del comando è quello di modificare al tempo corrente i tempi di ultimo accesso e ultima modifica.[14] Il comando prende varie opzioni e permette di modificare solo il tempo di ultimo accesso se usato con l'opzione `-a` o solo quello di ultima modifica, se usato con l'opzione `-m`, si può specificare una data con `-d` (per il formato si veda sez. 2.4.3) o prendere i valori da un altro file con l'opzione `-r`. I dettagli e l'elenco completo delle opzioni sono al solito sulla pagina di manuale, accessibile con `man touch`

1.2.2 L'architettura di un filesystem e le proprietà dei file

Come già accennato Linux (ed ogni sistema unix-like) organizza i dati che tiene su disco attraverso l'uso di un filesystem. Una delle caratteristiche di Linux rispetto agli altri Unix è quella di poter supportare, grazie al VFS, una enorme quantità di filesystem diversi, ognuno dei quali ha

[12] per una trattazione dettagliata di questo argomento si può consultare il capitolo 4.3.4 di [GaPiL].

[13] a meno che questa funzionalità non sia stata disabilitata, vedremo come farlo per un filesystem in sez. 5.1.3 e per un singolo file in sez. 1.4.4, per i kernel recenti inoltre di default questo tempo viene cambiato al massimo una volta al giorno.

[14] il tempo di ultimo cambiamento di stato viene modificato solo dal kernel, e non esiste una modalità immediata (che non sia quella di andare a sovrascrivere l'informazione con un accesso diretto al dispositivo del disco) per cambiarlo manualmente.

una sua particolare struttura e funzionalità proprie.[15] Per questo non entreremo nei dettagli di un filesystem specifico, ma daremo una descrizione a grandi linee che si adatta alle caratteristiche comuni di un filesystem generico usato su un sistema unix-like.

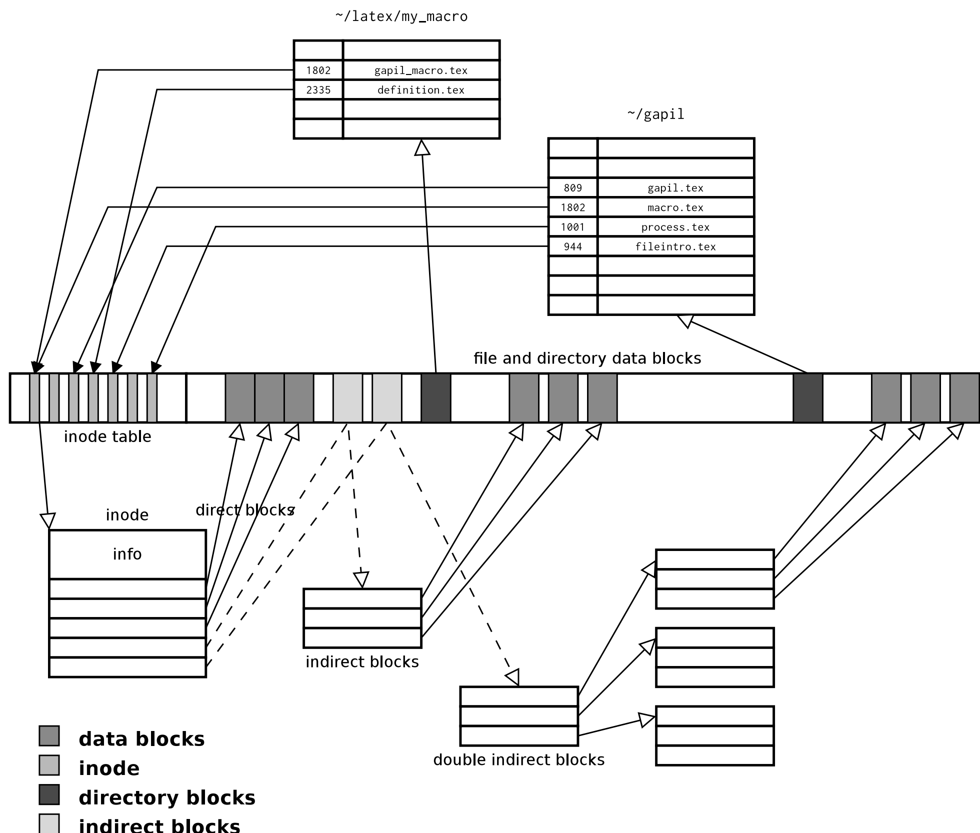


Figura 1.2: Strutturazione dei dati all'interno di un filesystem.

Se si va ad esaminare con maggiore dettaglio come è strutturata l'informazione all'interno di un singolo filesystem, possiamo esemplificare la situazione con uno schema come quello esposto in fig. 1.2, da cui si evidenziano alcune delle caratteristiche di base di un filesystem, sulle quali è bene porre attenzione visto che sono fondamentali per capire il funzionamento dei comandi che manipolano i file e le directory.

La struttura che identifica univocamente un singolo file all'interno di un filesystem è il cosiddetto *inode*: ciascun file è associato ad un *inode* in cui sono mantenute tutte le informazioni che lo riguardano, come il tipo (fra quelli di tab. 1.1), i permessi di accesso (vedi sez. 1.4.2), utente e gruppo proprietario, le dimensioni, i tempi, ed anche tutti i riferimenti ai settori del disco (i *blocchi fisici*) che contengono i dati; le informazioni che fornisce il comando `ls` provengono dall'*inode*.

[15]riprenderemo l'argomento della gestione di dischi e filesystem in sez. 5.1.

L'unica informazione relativa al file non contenuta nell'*inode* è il suo nome; infatti il nome di un file non è una proprietà del file, ma semplicemente una etichetta associata ad un *inode*. Le directory infatti non *contengono* i file, ma sono dei file speciali (di tipo *directory*, così che il kernel possa trattarle in maniera diversa) il cui contenuto è semplicemente una lista di nomi a ciascuno dei quali viene associato un numero di *inode* che identifica il file cui il nome fa riferimento.

Come mostrato in fig. 1.2 si possono avere più voci in directory diverse che puntano allo stesso *inode*. Questo introduce il concetto di *hard link* (detto anche *collegamento diretto*): due file che puntano allo stesso *inode* sono fisicamente lo stesso file, nessuna proprietà specifica, come permessi, tempi di accesso o contenuto permette di distinguerli, in quanto l'accesso avviene per entrambi attraverso lo stesso *inode*.

Siccome uno stesso *inode* può essere referenziato in più directory, un file può avere più nomi, anche completamente scorrelati fra loro. Per questo ogni *inode* mantiene un contatore che indica il numero di riferimenti che gli sono stati fatti, il cosiddetto *link count*. Questo numero viene mostrato, nell'esempio del precedente risultato di `ls -l`, dal valore riportato nella seconda colonna della stampa dei risultati, e ci permette di dire se un file ha degli *hard link*, anche se non indica dove sono.

Il comando generico che permette di creare dei *link* è `ln` che prende come argomenti il file originale ed il nome del nuovo link. La differenza rispetto a Windows o MacOS è che in un sistema unix-like i collegamenti per i file sono di due tipi, oltre agli *hard link* appena illustrati infatti esistono anche i cosiddetti *collegamenti simbolici*, o *symbolic link*, che sono quelli effettivamente più simili ai collegamenti di Windows o agli alias del MacOS, come il file `link` mostrato in precedenza.

Per creare un *hard link* basta usare direttamente il comando `ln`, (il cui nome deriva da ***link** file*) che di default crea questo tipo di collegamento. Così possiamo creare un nuovo file `hardlink` per il file `file` visto in precedenza usando il comando:

```
piccardi@oppish:~/filetypes$ ln file hardlink
```

e se riesaminiamo la lista dei file otterremo che:

```
piccardi@oppish:~/filetypes$ ls -l
total 1
brw-r--r--    1 root     root       1,   2 Jul  8 14:48 block
crw-r--r--    1 root     root       1,   2 Jul  8 14:48 char
drwxr-xr-x    2 piccardi piccardi       48 Jul  8 14:24 dir
prw-r--r--    1 piccardi piccardi        0 Jul  8 14:24 fifo
-rw-r--r--    2 piccardi piccardi        0 Jul  8 14:24 file
-rw-r--r--    2 piccardi piccardi        0 Jul  8 14:24 hardlink
lrwxrwxrwx    1 piccardi piccardi        4 Jul  8 14:25 link -> file
```

Si noti come adesso la seconda colonna dell'output del comando mostri sia per `file` che per `hardlink` un valore pari a due, come ci si aspetta dato che entrambi fanno riferimento allo stesso *inode* che pertanto ha due collegamenti. Usando l'opzione `-i` di `ls` possiamo anche stampare per ciascun file il rispettivo numero di *inode*, ottenendo:

```
piccardi@oppish:~/filetypes$ ls -li
total 1
   2118 brw-r--r--    1 root     root       1,   2 Jul  8 14:48 block
   2120 crw-r--r--    1 root     root       1,   2 Jul  8 14:48 char
     15 drwxr-xr-x    2 piccardi piccardi       48 Jul  8 14:24 dir
```

```
2115 prw-r--r--    1 piccardi piccardi       0 Jul  8 14:24 fifo
2117 -rw-r--r--    2 piccardi piccardi       0 Jul  8 14:24 file
2117 -rw-r--r--    2 piccardi piccardi       0 Jul  8 14:24 hardlink
2116 lrwxrwxrwx    1 piccardi piccardi       4 Jul  8 14:25 link -> file
```

e si può verificare come effettivamente `file` e `hardlink` abbiano lo stesso numero di *inode*, che nel caso è 2117.

Un limite che si ha con gli *hard link* è dovuto al fatto che si può fare riferimento solo ad un *inode* presente nello stesso filesystem della directory. Questo avviene perché le directory contengono semplicemente l'associazione fra un nome ed un numero di *inode* ed ogni filesystem numera progressivamente ed in maniera indipendente i propri *inode*, per cui su un altro filesystem lo stesso numero di *inode* identificherà un file diverso. Questo limita l'uso degli *hard link* solo a file residenti sul filesystem corrente, ed infatti il comando `ln` darà un errore se si cerca di creare un *hard link* ad un file posto in un altro filesystem.

Per superare questa limitazione sono stati introdotti i cosiddetti *collegamenti simbolici* o *symbolic link*, che vengono creati usando l'opzione `-s` del comando `ln`; ad esempio si è creato il collegamento simbolico `link` mostrato nell'esempio precedente con il comando:

```
piccardi@oppish:~/filetypes$ ln -s file link
```

In questo caso viene creato un nuovo file, di tipo *symbolic link*, che avrà un suo diverso *inode* (come risulta nell'output di `ls -li` appena mostrato) ed il cui contenuto è il percorso (torneremo sulla notazione che esprime i *pathname* dei file fra poco) da fare per arrivare al file a cui esso fa riferimento, che a questo punto può essere in qualsiasi altro filesystem. È compito del kernel far sì che quando si usa un link simbolico si vada poi ad operare sul file che questo ci indica,[16] ed è questo il motivo per cui per i link simbolici esiste un apposito tipo di file, come riportato in tab. 1.1.

Oltre a `-s` il comando `ln` prevede una serie di altre opzioni, le principali delle quali sono riportate in tab. 1.4. La lista completa delle opzioni, e la relativa documentazione, è disponibile nella pagina di manuale, accessibile attraverso il comando `man ln`.

Opzione	**Significato**
`-s`	crea un collegamento simbolico.
`-f`	forza la sovrascrittura del nuovo file se esso esiste già.
`-i`	richiede conferma in caso di sovrascrittura.
`-d`	crea un *hard link* ad una directory (in Linux questa non è usabile).
`-b`	esegue un backup della destinazione se questa esiste già.

Tabella 1.4: Principali opzioni del comando `ln`.

Una seconda caratteristica dei link simbolici è la possibilità di creare dei collegamenti anche per delle directory. Questa capacità infatti, sebbene teoricamente possibile anche per gli *hard link*, in Linux non è supportata per la sua pericolosità, è possibile infatti creare dei *link loop* se si commette l'errore di creare un collegamento alla directory che contiene il collegamento stesso; con un collegamento simbolico questo errore può essere corretto in quanto la cancellazione del

[16] una trattazione dettagliata su come vengono gestite dal kernel le operazioni sui link simbolici si può trovare in sez. 4.2.1 di [GaPiL].

link simbolico rimuove quest'ultimo, e non il file referenziato, ma con un *hard link* non è più possibile fare questa distinzione e la rimozione diventa impossibile.

La possibilità di creare dei collegamenti alle directory tuttavia è estremamente utile, infatti qualora si voglia accedere ai file in una directory attraverso un percorso diverso, ad esempio a causa di un programma che cerca dei file di configurazione in una locazione diversa da quella usuale, piuttosto che dover spostare tutti i file basterà creare un collegamento simbolico e si sarà risolto il problema.

Oltre agli *hard link* la struttura di un filesystem unix-like ha ulteriori conseguenze non immediate da capire per chi proviene da sistemi operativi diversi. La presenza degli *hard link* e l'uso degli *inode* nelle directory infatti comporta anche una modalità diversa nella cancellazione dei file e nello spostamento degli stessi.

Il comando per la cancellazione di un file è `rm` (il cui nome deriva da ***rem**ove file*), ma la funzione usata dal sistema per effettuare questo compito si chiama in realtà *`unlink`* ed essa, come ci dice il nome, non cancella affatto i dati del file, ma si limita ad eliminare la relativa voce da una directory e decrementare il numero di riferimenti presenti nell'*inode*.

Solo quando il numero di riferimenti ad un *inode* si annulla, i dati del file vengono effettivamente rimossi dal disco dal kernel. In realtà oltre ai riferimenti mostrati da `ls` il kernel mantiene anche un'altra lista di riferimenti, per ciascun processo che sta accedendo al file, per cui anche se si cancella un file da una directory e non restano altri *hard link* al relativo *inode*, ma resta attivo un processo che lo utilizza, lo spazio disco non verrà rilasciato, e fintanto che il processo non avrà finito i dati resteranno, anche se solo per lui, disponibili.[17]

In realtà anche questo è vero solo parzialmente, con la sua cancellazione i dati di un file in realtà non vengono toccati, viene solo dichiarato come disponibile lo spazio da loro occupato, che così può essere riutilizzato. Questo comporta che è possibile, se nessuno ha usato tale spazio, trovare ancora sul disco i suddetti dati con degli appositi programmi di recupero. Per questo quando si vuole essere davvero sicuri che i dati siano cancellati si deve utilizzare al posto di `rm` un comando come `shred` che provvede anche ad una opportuna sovrascrittura che renda sostanzialmente impossibile recuperare il precedente contenuto del file (per i dettagli si consulti la pagina di manuale con `man shred`).

Il comando `rm` prende come argomenti una lista di file da cancellare; se si usa l'opzione `-i` il comando chiede di confermare la cancellazione, mentre con l'opzione `-f` si annulla ogni precedente `-i` ed inoltre non vengono stampati errori per file non esistenti. Infine l'opzione `-R` (o `-r`) permette la cancellazione ricorsiva di una directory e di tutto il suo contenuto, ed è pertanto da usare con estrema attenzione, specie se abbinata con `-f`.[18] La lista completa delle opzioni è riportata nella pagina di manuale, accessibile con il comando `man rm`.

Come accennato la struttura di un filesystem unix-like comporta anche una diversa concezione dell'operazione di spostamento dei file, che nel caso più semplice è semplicemente equivalente a quella di cambiare il nome del file. Il comando per compiere questa operazione infatti è unico e si chiama `mv`, il cui nome deriva da ***mov**e file*.

[17] per approfondire questo aspetto dell'architettura dei file per un sistema operativo unix-like, si può consultare il capitolo 6 di [GaPiL].

[18] uno dei sistemi più efficaci per distruggere una installazione è un `rm -fR` eseguito come amministratore nel posto sbagliato, e anche se come utente non si può danneggiare il sistema, è comunque molto semplice spazzare via tutti i propri dati.

Infatti fintanto che si "*sposta*" un file da una directory ad un altra senza cambiare filesystem, non c'è nessuna necessità di spostare il contenuto del file e basta semplicemente che sia creata una nuova voce per l'*inode* in questione rimuovendo al contempo la vecchia: esattamente la stessa cosa che avviene quando gli si cambia nome (nel qual caso l'operazione viene effettuata all'interno della stessa directory). Qualora invece si debba effettuare lo spostamento ad una directory su un filesystem diverso diventa necessario prima copiare il contenuto e poi cancellare l'originale (si ricordi infatti che in questo caso si avrà a che fare con un *inode* diverso, situato in un filesystem diverso).

Il comando `mv` ha due forme, può prendere cioè come argomenti o due nomi di file o una lista di file seguita da una directory. Nel primo caso rinomina il primo file nel secondo, cancellando quest'ultimo qualora esista già, nel secondo caso sposta tutti i file della lista nella directory passata come ultimo argomento, sempre cancellando eventuali file con lo stesso nome in essa presenti.

Dato che anche le directory sono file, il comando opera anche su queste usando la stessa sintassi ma con qualche piccola differenza; se ad esempio si cerca di rinominare una directory col nome di un file già presente si avrà un errore, e se si cerca di rinominare una directory col nome di un'altra si otterrà invece uno spostamento dentro quest'ultima. Le principali opzioni del comando sono riportate in tab. 1.5, l'elenco completo è riportato nella pagina di manuale, accessibile con il comando `man mv`.

Opzione	**Significato**
`-i`	richiede conferma in caso di sovrascrittura.
`-f`	forza la sovrascrittura del nuovo file se esso esiste già.
`-u`	esegue lo spostamento solo se la destinazione è più vecchia della sorgente (o non esiste).
`-b`	esegue un backup della destinazione se questa esiste già.

Tabella 1.5: Principali opzioni del comando `mv`.

Dato che il comando si limita a cambiare una voce associata ad un numero di *inode* all'interno di una directory, fintanto che lo spostamento avviene all'interno dello stesso filesystem, i tempi dei file non vengono modificati dall'uso di `mv`. Quando però lo spostamento avviene fra filesystem diversi viene copiato il contenuto e cancellato il file originario, pertanto in teoria dovrebbero risultare modificati anche i tempi di ultimo accesso e modifica. In realtà il comando provvede a ripristinare questi tempi (come le altre caratteristiche del file) al valore del file originario, ma non può fare nulla per ripristinare il tempo di ultimo cambiamento.[19] Pertanto in quel caso si potrà notare, usando `ls -lc`, che questo è cambiato e corrisponde al momento dello spostamento.

Qualora invece si voglia duplicare un file il comando da usare è `cp`, il cui nome deriva da ***copy** file*. Come per `mv` il comando può prendere come argomenti o due nomi di file o una lista di file seguita da una directory; nel primo caso effettua una copia del primo file sul secondo, nel secondo copia tutti file della lista nella directory specificata.

[19] il kernel fornisce delle *system call* che permettono di cambiare i tempi di ultimo accesso e modifica di un file, che il comando `mv` può usare per ripristinare i tempi precedenti, ma non ne esistono per cambiare il tempo di ultimo cambiamento, che corrisponderà pertanto al momento in cui il nuovo file è stato creato; questa è una misura di sicurezza che permette sempre di verificare se un file è stato modificato, anche se si cerca di nascondere le modifiche.

Dato che il comando funziona copiando il contenuto di un file su un nuovo file creato per l'occasione, i tempi di ultima modifica, accesso e cambiamento di quest'ultimo corrisponderanno al momento in cui si è eseguita l'operazione. Inoltre il file sarà creato con i permessi standard dell'utente che ha lanciato il comando, che risulterà essere anche il suo proprietario. Se si vogliono preservare invece le caratteristiche del file originale occorrerà usare l'opzione `-p`.[20]

Opzione	Significato
`-f`	forza la sovrascrittura della destinazione se essa esiste già.
`-i`	richiede conferma in caso di sovrascrittura.
`-p`	preserva tempi, permessi e proprietari del file.
`-l`	crea degli *hard link* al posto delle copie.
`-s`	crea dei collegamenti simbolici al posto delle copie.
`-d`	copia il link simbolico invece del file da esso indicato.
`-r`	copia ricorsivamente tutto il contenuto di una directory.
`-R`	identico a `-r`.
`-a`	combina le opzioni `-dpR`.
`-L`	segue sempre i collegamenti simbolici.
`-b`	esegue un backup della destinazione se questa esiste già.
`-u`	esegue la copia solo se la destinazione è più vecchia della sorgente (o non esiste).

Tabella 1.6: Principali opzioni del comando `cp`.

Si tenga presente poi che nel caso di collegamenti simbolici il comando copia il file indicato tramite il link,[21] se invece si vuole copiare il link stesso occorrerà usare l'opzione `-d`. Il comando permette inoltre di creare degli *hard link* invece che delle copie usando l'opzione `-l`, e dei collegamenti simbolici usando l'opzione `-s`. Una lista delle principali opzioni è riportata in tab. 1.6, l'elenco completo è riportato nella pagina di manuale, accessibile attraverso il comando `man cp`.

Fino ad ora nel descrivere gli argomenti da dare ai vari comandi che abbiamo trattato, si è parlato dei nomi dei file o delle directory senza entrare troppo nei dettagli su quale fosse il formato in cui questi vengono espressi. Negli esempi infatti si sono specificati dei semplici nomi, ma questo dava per scontati alcuni concetti che in realtà non lo sono affatto.

La convenzione usata in tutti i sistemi unix-like è che i nomi dei file sono indicati con un *pathname* o *percorso*, che descrive il cammino che occorre fare nell'albero dei file per raggiungere il file passando attraverso le varie directory, dove i nomi delle directory sono separati fra loro da delle barre (il carattere "`/`").

Il percorso può essere indicato (vedi tab. 1.7) in maniera assoluta, cioè partendo dalla directory radice (torneremo sulla struttura dell'albero dei file e sul concetto di radice in sez. 1.2.3) ed indicando tutte le directory che si devono attraversare, o in maniera relativa, partendo dalla cosiddetta *directory di lavoro corrente*. Nel primo caso il *pathname* inizierà con una "`/`", mentre nel secondo no.

Finora, specificando solo dei nomi semplici, abbiamo sempre fatto l'assunto di operare appunto nella *directory di lavoro corrente*. Questa directory è una proprietà specifica di ogni processo, che viene ereditata dal padre alla sua creazione (questo argomento è trattato in sez. 1.3.2) ed

[20] però per poterlo fare in maniera completa, conservando il proprietario originale, per le ragioni che vedremo in sez. 1.4.3 occorre eseguire il comando come amministratore.

[21] questo comportamento non viene seguito in caso di copia ricorsiva, se lo si vuole abilitare occorre usare l'opzione `-L`.

Esempio	**Formato**
`/home/piccardi/GaPiL/GaPiL.tex`	assoluto
`GaPiL/GaPiL.tex`	relativo

Tabella 1.7: Formato dei *pathname* assoluti e relativi.

indica appunto la directory a partire dalla quale vengono risolti, per quel processo, i *pathname* relativi. Questo vuol dire che si possono indicare i file presenti in quella directory direttamente, senza doverne specificare il *pathname* completo a partire dalla radice.

Quando si entra nel sistema la directory di lavoro corrisponde alla cosiddetta *home* dell'utente. Tradizionalmente infatti ogni utente del sistema ha una sua directory personale nella quale può tenere i propri file, che viene chiamata appunto "*casa*" e quando entra nel sistema viene posizionato in tale directory; ritorneremo sull'argomento più avanti in sez. 1.2.3.

La directory di lavoro può essere cambiata con il comando `cd` (il cui nome deriva ***c**hange **d**irectory*) seguito dal *pathname* della directory in cui ci si vuole spostare. Per sapere quale è la si può stampare a video con il comando `pwd`, il cui nome deriva da ***p**rint **w**ork **d**irectory*.

Si tenga presente poi che ciascuna directory contiene sempre almeno due voci: la directory "`.`" che fa riferimento a se stessa, e che usata all'inizio di un *pathname* indica la directory di lavoro corrente, e la directory "`..`", che fa riferimento alla directory in cui l'attuale è contenuta, e che all'inizio di un *pathname* indica la directory sovrastante quella corrente. In questo modo, anche con dei *pathname* relativi, si possono fare riferimenti a directory poste in sezioni diverse dell'albero dei file, risalendo lo stesso con l'uso della directory "`..`" che permette di arrivare fino alla radice dell'albero dei file.[22] Si noti anche come entrambe queste due voci, dato che iniziano per "`.`" siano "*invisibili*".

Inoltre, come vedremo meglio in sez. 2.1.3, la shell quando deve passare dei *pathname* ai comandi che operano su file e directory (come `cd`, `cp`, ecc.) riconosce alcuni caratteri speciali, ad esempio il carattere "`~`" viene usato per indicare la home dell'utente corrente, mentre con `~username` si indica la home dell'utente `username`; `cd` riconosce inoltre come argomento il carattere "`-`" che viene utilizzato per richiedere il ritorno alla precedente directory di lavoro.

Come accennato nella sezione precedente anche le directory sono dei file, anche se vengono trattate dal kernel come file speciali il cui compito è quello di contenere elenchi di nomi di altri file. Per la creazione di una directory è previsto un comando apposito, `mkdir`. Questo è necessario perché alla creazione della directory devono essere create al suo interno anche le directory "`.`" e "`..`", che sono sempre presenti.[23]

Il comando `mkdir` crea la o le directory passate come argomento; queste dovranno essere specificate con quello che sarà il loro *pathname*, sia in forma assoluta o relativa. Perché il comando abbia successo ovviamente la parte di percorso che deve contenere la directory che si vuole creare deve esistere, se non è così si può forzare la creazione di tutte le directory indicate nel percorso specificato con l'opzione `-p`, se ne possono anche impostare i permessi iniziali (torneremo su questo argomento in sez. 1.4.3) con l'opzione `-m`.

Come per la creazione, è necessario un apposito comando, `rmdir`, anche per la rimozione di una directory, dato che `rm`, a meno di non usare l'opzione `-R` che però cancella anche tutto il

[22] per la quale la directory "`..`" fa riferimento a se stessa, non essendovi niente al di sopra.

[23] è compito della omonima *system call* che effettua la creazione di una nuova directory far sì che esse siano sempre presenti.

contenuto, rimuove solo i file. Di nuovo il comando prende il nome di una o più directory vuote da cancellare. Se le directory non sono vuote, dove vuote significa che non ci deve essere niente a parte le due directory "`.`" e "`..`" il comando fallisce. Anche in questo caso si può usare l'opzione `-p` che cancella tutto un percorso di directory (che comunque devono essere tutte vuote).

1.2.3 L'albero dei file ed il *Filesystem Hierarchy Standard*

Una delle caratteristiche peculiari di un sistema unix-like è che l'albero delle directory è unico; non esistono cioè i vari dischi (o volumi) che si possono trovare in altri sistemi, come su Windows, sul MacOS o sul VMS. All'avvio il kernel *monta*[24] quella che si chiama la *directory radice* (o *root directory*) dell'albero, che viene indicata con "`/`", tutti i restanti dischi, il CDROM e qualunque altro dispositivo contenente file, verranno poi *montati* (vedi sez. 5.1.3) successivamente in opportune sotto-directory della radice.

Come per il processo iniziale `init`, che non è figlio di nessun altro processo e viene lanciato direttamente dal kernel, anche la directory radice non è contenuta in nessuna altra directory e, come accennato in sez. 1.1.1, viene montata direttamente dal kernel in fase di avvio. Per questo motivo la directory radice viene ad assumere un ruolo particolare, ed il filesystem che la supporta deve contenere tutti i programmi di sistema necessari all'avvio, compreso `init` o eventuali sostituti.

Un esempio di questa struttura ad albero, che al contempo ci mostra anche i contenuti delle directory principali, può essere ottenuto con il comando `tree`. Se chiamato senza parametri questo comando mostra l'albero completo a partire dalla directory corrente, scendendo in tutte le directory sottostanti; usando l'opzione `-L` si può specificare il numero massimo di livelli a cui scendere, per cui andando su `/` avremo qualcosa del tipo:

```
piccardi@hain:~$ tree --charset=ASCII -L 2 /
/
|-- bin
|   |-- bash
...
|-- boot
|   |-- config-3.2.0-1-amd64
...
|-- dev
|   |-- autofs
...
|-- etc
|   |-- acpi
...
|-- home
|   `-- piccardi
|-- lib
|   |-- discover
...
|-- lib32
|   |-- ld-2.13.so
...
```

[24] come accennato l'operazione che rende visibili ai processi i file contenuti all'interno di un filesystem facendoli comparire all'interno nell'albero dei file viene chiamata *montaggio* il filesystem, torneremo sull'argomento in sez. 5.1.3.

```
|-- lost+found
|-- media
|-- mnt
|-- opt
|-- proc
|   |-- 1
...
|-- root
|-- run
|   |-- atd.pid
...
|-- sbin
|   |-- acpi_available
...
|-- selinux
|-- srv
|-- sys
|   |-- block
...
|-- tmp
|   |-- ssh-LjWlESaD8958
...
|-- usr
|   |-- bin
|   |-- games
|   |-- include
|   |-- lib
|   |-- lib32
|   |-- local
|   |-- sbin
|   |-- share
|   `-- src
|-- var
|   |-- backups
|   |-- cache
|   |-- games
|   |-- lib
|   |-- local
|   |-- lock -> /run/lock
|   |-- log
|   |-- mail
|   |-- opt
|   |-- run -> /run
|   |-- spool
|   `-- tmp
`-- vmlinuz -> boot/vmlinuz-3.2.0-1-amd64
```

e questo ci mostra il contenuto sommario dei primi due livelli dell'albero, con un esempio dei file e delle sottodirectory presenti in una versione di test di Debian (Wheezy).

L'organizzazione dell'albero delle directory è standardizzata in maniera molto accurata da un documento che si chiama *Filesystem Hierarchy Standard* (abbreviato in FHS), preso come riferimento da tutte le distribuzioni.[25] Lo standard descrive in dettaglio la struttura dell'albero

[25]al momento dell'ultima revisione di questo testo (Febbraio 2013) la versione corrente è la 2.3, rilasciata nel 2004 e parte delle specifiche LSB (*Linux Standard Base*), ma è in corso di definizione la versione 3.0, con varie novità come `/run`.

delle directory e il relativo contenuto, prevedendo una divisione molto rigorosa che permette una notevole uniformità anche fra distribuzioni diverse; si organizzano così in maniera meticolosa ed ordinata dati, programmi, file di configurazione, documentazione, file degli utenti, ecc.

Directory	Contenuto
/bin	comandi essenziali.
/boot	file statici necessari al *bootloader*.
/dev	file di dispositivo.
/etc	file di configurazione della macchina.
/lib	librerie essenziali e moduli del kernel.
/media	*mount point* per dispositivi rimovibili.
/mnt	*mount point* montaggi temporanei.
/opt	pacchetti software addizionali.
/run	dati di *run-time* volatili.
/sbin	comandi di sistema essenziali.
/srv	dati per i servizi forniti dal sistema.
/tmp	file temporanei.
/usr	gerarchia secondaria.
/var	dati variabili.

Tabella 1.8: Sottodirectory di / obbligatorie per qualunque sistema.

In particolare le directory vengono suddivise sulla base di alcuni criteri fondamentali. Il primo è quello della possibilità di contenere file il cui contenuto può essere modificato (nel qual caso il filesystem che le contiene deve essere montato in lettura/scrittura) o meno (nel qual caso il filesystem può essere montato in sola lettura).

Il secondo è quello della possibilità di contenere file come i programmi di sistema che possono essere condivisi (ad esempio utilizzando un filesystem di rete) fra più stazioni di lavoro o file che invece sono locali e specifici alla macchina in questione.

Il terzo criterio è quello di contenere o meno comandi o file (configurazioni e file di dispositivo) che sono necessari all'avvio del sistema, e che pertanto devono essere situati sul filesystem usato per la directory radice, dato che essi non sarebbero disponibili se posti in filesystem diversi che possono essere montati solo dopo che il sistema è partito.

Lo standard prevede che debbano essere necessariamente presenti le sottodirectory di / specificate in tab. 1.8, mentre quelle di tab. 1.9 sono obbligatorie soltanto qualora si siano installati i sottosistemi a cui essi fanno riferimento (utenti, /proc filesystem, diversi formati binari).[26]

Un elenco delle specifiche delle caratteristiche e del contenuto di ciascuna delle sottodirectory di / è riportato di seguito; per alcune di esse, come /usr e /var, sono previste delle ulteriori sottogerarchie che definiscono ulteriori dettagli dell'organizzazione dei file. Si è aggiunto all'elenco anche la directory /run, che pur non essendo prevista dal FHS è stata adottata da tutte le distribuzioni più recenti.

/bin Contiene i comandi essenziali del sistema (usati sia dall'amministratore che dagli utenti, come ls), che devono essere disponibili anche quando non ci sono altri filesystem montati oltre la radice, ad esempio all'avvio o quando si è in *single user mode* (vedi

[26] le eventuali /lib<qual> contengono le versioni delle librerie di sistema in formati binari diversi; le /lib alternative sono state usate al tempo della transizione dei programmi dal formato *a.out* ad *ELF*, oggi sono usate principalmente per quei sistemi (come gli AMD-64) che supportano diversi formati binari (32 o 64 bit).

Directory	Contenuto
`/lib<qual>`	librerie in formati alternativi.
`/home`	home directory degli utenti.
`/root`	home directory dell'amministratore.
`/proc`	filesystem virtuale con informazioni su processi e caratteristiche del sistema.
`/sys`	filesystem virtuale con informazioni su driver e dispositivi (vedi sez. 5.4.5).

Tabella 1.9: Sottodirectory di `/` obbligatorie solo in presenza dei relativi sottosistemi.

sez. 5.3.5). Non deve avere sottodirectory e non può stare su un filesystem diverso da quello della radice.

`/boot` Contiene tutti i file necessari al procedimento di boot (immagini del kernel, del *bootloader*, *ramdisk*, ecc.) eccetto i file di configurazione ed i programmi per l'impostazione del procedimento stesso, che vanno in `/sbin`. Può stare su qualunque filesystem purché visibile dal *bootloader* (vedi sez. 5.3.1).

`/dev` Contiene i file di dispositivo, che permettono l'accesso alle periferiche. Originariamente doveva stare sullo stesso filesystem della radice, dato che i file di dispositivo sono necessari alla procedura di avvio. Oggi il contenuto è nella maggior parte dei casi generato dinamicamente ed è montata su un filesystem temporaneo[27] che viene popolato in fase di avvio tramite `udev` (vedi sez. 5.4.5).

`/etc` Contiene i file di configurazione del sistema e gli script[28] di avvio. Non deve contenere programmi binari e non può stare su un filesystem diverso da quello della radice. I file possono essere raggruppati a loro volta in directory; lo standard prevede solo che, qualora siano installati, siano presenti le directory `/etc/opt` (per i pacchetti opzionali), `/etc/X11` (per la configurazione di *X Window*, vedi sez. 3.3) e `/etc/sgml` e `/etc/xml` (per la configurazione di SGML e XML).

`/home` Contiene le *home directory* degli utenti, la sola parte del filesystem (eccetto `/tmp` e `/var/tmp`) su cui gli utenti hanno diritto di scrittura, che viene usata per mantenere i loro file personali. Può essere montata su qualunque filesystem.

`/lib` Contiene le librerie condivise essenziali, usate dai programmi di `/bin` e `/sbin`, e deve essere sullo stesso filesystem della radice. Qualora sia stato installato un kernel modulare (vedi sez. 5.2.5) i moduli devono essere installati in `/lib/modules`.

`/media` Contiene i *mount point* (vedi sez. 5.1.3) per i dispositivi rimovibili, come CDROM, floppy, chiavette USB, dischi USB o Firewire, ecc. Nelle distribuzioni meno recenti i floppy ed i CDROM hanno delle directory dedicate, ma con l'introduzione di meccanismi di rilevazione automatica dei dispositivi rimovibili (vedi sez. 5.4.5) è stata definita

[27] si intende con questo un filesystem di tipo `tmpfs` (vedi tab. 5.2) il cui contenuto è mantenuto in memoria.

[28] gli *script*, su cui torneremo in sez. 2.1.5, sono un po' gli equivalenti in ambito Unix (come potrebbe esserlo una Ferrari in confronto ad una 500) dei file `.bat` del DOS, una lista di comandi messi in un file (in realtà si tratta di un vero linguaggio di programmazione) e fatti eseguire automaticamente.

questa directory all'interno della quale vengono create le opportune sotto-directory su cui poi essi vengono montati un maniera automatica.

/mnt Contiene i *mount point* (vedi sez. 5.1.3) per i montaggi temporanei ad uso dell'amministratore di sistema; i filesystem di periferiche permanenti con supporti rimovibili come i floppy o il CDROM che prima venivano tenuti sia in questa directory che direttamente sotto `/` devono essere spostati sotto `/media`. Normalmente è vuota e deve essere creata direttamente sotto la radice.

/opt Contiene eventuali pacchetti software aggiuntivi. Può essere su qualunque filesystem. Un pacchetto deve installarsi nella directory `/opt/package` dove *package* è il nome del pacchetto. All'amministratore è riservato l'uso di alcune directory opzionali: `/opt/bin`, `/opt/doc`, `/opt/include`, `/opt/info`, `/opt/lib` e `/opt/man`. File variabili attinenti ai suddetti pacchetti devono essere installati in `/var/opt` ed i file di configurazione in `/etc/opt`, nessun file attinente ai pacchetti deve essere installato al di fuori di queste directory.

/proc È il *mount point* standard del filesystem virtuale `proc`. Questo è un filesystem speciale che permette di accedere a tutta una serie di variabili interne al kernel (relative a parametri e impostazioni di tutti tipi) con l'interfaccia dei file. Così se si vogliono informazioni sugli interrupt ed i canali di DMA (vedi sez. 5.4.1) utilizzati dal sistema si potranno leggere i file `/proc/interrupts` e `/proc/dma`, mentre si potranno impostare varie caratteristiche del sistema scrivendo nei file sotto `/proc/sys` (vedi sez. 3.1.3).

/root È la *home directory* dell'amministratore. Di norma la si mantiene nello stesso filesystem della radice; il suo uso è opzionale ma questa è la collocazione consigliata.

/run Presente solo nelle distribuzioni più recenti, viene utilizzata per mettere a disposizione fin dall'avvio una directory su cui viene montato un filesystem temporaneo e su cui sono mantenuti i cosiddetti dati di *run-time* volatili (come i file che registrano i PID dei programmi di servizio, i file di lock ed altri file il cui utilizzo attiene principalmente all'esecuzione di programmi) di cui non serve il mantenimento attraverso un riavvio. La directory consente di avere un posto dove anche i programmi lanciati all'avvio possono mantenere i loro dati, dato che in generale `/var/run` e `/var/lock` possono non essere disponibili quando `/var` viene mantenuta su un filesystem separato. Consente anche di unificare in un'unica directory e su un unico filesystem temporaneo tutti questi dati, dato che se viene utilizzata il contenuto di `/var/run` e `/var/lock` viene riportato al suo interno.

/sbin Contiene i programmi essenziali ad uso dell'amministratore del sistema (come `init` o `fsck`). Deve stare sullo stesso filesystem della radice. Vanno messi in questa directory solo i programmi essenziali per l'avvio del sistema, il recupero e la manutenzione dei filesystem.

/srv È stata introdotta per mantenervi i dati relativi ai vari servizi che possono essere stati installati su una macchina (come ad esempio le pagine servite da un web server) che

in precedenza venivano installati direttamente sotto `/var`.[29] Non ne è definita una ulteriore suddivisione, ma in genere si tende a creare una sottodirectory per ciascun servizio (ad esempio `www`, `svn`, ecc.).

`/tmp` La directory viene usata per mantenere file temporanei. Viene cancellata ad ogni riavvio, ed i programmi non devono assumere che i file siano mantenuti fra due esecuzioni successive.

`/usr` È la directory principale che contiene tutti i programmi, i file ed i dati non variabili, che possono anche essere condivisi fra più stazioni di lavoro. Può essere montata su un filesystem separato rispetto a `/` e può essere montata in sola lettura, anche se questa scelta potrebbe rendere problematico l'uso di alcuni ambienti. Prevede una ulteriore gerarchia di directory in cui i vari file vengono organizzati; lo standard richiede obbligatoriamente le seguenti:

`bin` contiene i programmi usati dall'utente installati direttamente dal sistema (o dalla distribuzione originale). Non può essere ulteriormente suddivisa;

`include` contiene tutti i file di dichiarazione delle funzioni di libreria usati dal compilatore e dai programmi C e C++;

`lib` contiene le librerie, file oggetto e binari relativi ai programmi di `bin` e `sbin`. Può contenere sottodirectory con tutti i file relativi a singoli programmi;

`local` contiene una replica della gerarchia di `/usr` dedicata ai file installati localmente dall'amministratore. In genere qui vengono installati i programmi compilati dai sorgenti e tutto quello che non fa parte della distribuzione ufficiale;

`sbin` contiene i programmi per la gestione di sistema ad uso dell'amministratore non essenziali all'avvio;

`share` contiene una gerarchia in cui sono organizzati tutti i file e i dati che non dipendono dall'architettura hardware: sono sempre obbligatori `man` per le pagine di manuale, e `misc`; se installati vi si possono trovare anche una serie di altre directory come: `dict` per i dizionari, `doc` per la documentazione, `games` per i dati statici dei giochi, `info` per i file del relativo sistema di help (vedi sez. 2.4.2), `terminfo` per il database con le informazioni sui terminali, `zoneinfo` per i dati relativi ai fusi orari (per l'elenco completo si consulti il sito citato). Deve essere posto dentro `misc` tutto quello che non viene classificato nelle altre directory opzionali;

mentre sono obbligatorie solo se i relativi pacchetti sono installati, le seguenti directory:

`X11R6` conteneva la gerarchia dei file relativi ad *X Window* (vedi sez. 3.3), dalla versione 7.0 di *X.org* è stata rimossa;

`games` contiene i binari dei giochi;

`src` contiene i sorgenti dei pacchetti, per riferimenti, ed è opzionale;

[29] è stata introdotta con la versione 2.3 del FHS, ma il suo uso non si è molto diffuso.

/var Contiene i file variabili: le directory di spool, i file di log, i dati transienti e temporanei, in modo che `/usr` possa essere montata in sola lettura. È preferibile montarla in un filesystem separato; alcune directory non possono essere condivise. Anche in questo caso i file sono organizzati in una ulteriore gerarchia standardizzata che prevede la presenza delle seguenti sottodirectory:

- **cache** dati di appoggio e memorizzazione temporanea per le applicazioni;
- **lib** informazioni di stato e dati delle applicazioni;
- **local** dati variabili relativi ai pacchetti di `/usr/local`;
- **lock** file di lock, se si usa `/run` è un collegamento simbolico a `/run/lock`;
- **log** file di log delle applicazioni (vedi 3.2.3);
- **opt** file variabili per i pacchetti di `/opt`;
- **spool** directory per le code contenenti i dati in transito delle applicazioni (lavori di stampa, posta elettronica, ecc.);
- **tmp** file temporanei non cancellati al riavvio del sistema;

a queste si aggiungono tradizionalmente altre directory, come `/var/backups`, `/var/cron`, ecc. non indicate dallo standard.

In fig. 1.3 è riportata una rappresentazione grafica della struttura generale delle directory prevista dal FHS, anche se si è mostrata solo una parte delle directory previste. I dettagli completi sulla struttura, così come le specifiche relative ad i contenuti delle varie directory, possono essere reperiti sul documento ufficiale di definizione del FHS, disponibile all'indirizzo `http://www.pathname.com/fhs/`.

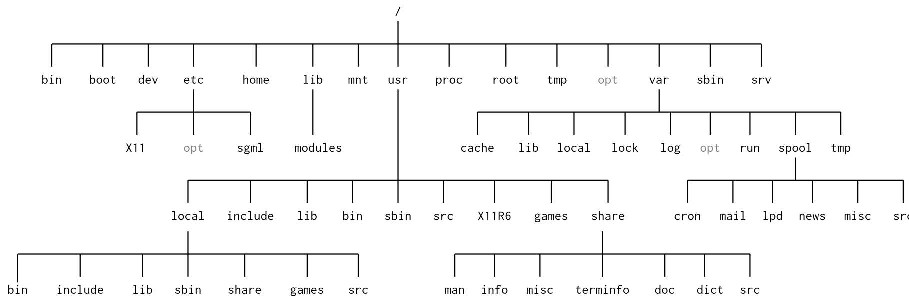


Figura 1.3: Struttura tipica delle directory, secondo il *Filesystem Hierarchy Standard*.

L'importanza del *Filesystem Hierarchy Standard* diventa evidente quando si vanno ad esaminare le strategie di partizionamento dei dischi (torneremo sull'argomento in sez. 5.1.2). In tal caso infatti occorrerà stabilire quali directory dovranno restare sul filesystem usato come radice, e quali altre directory si potranno disporre su altre partizioni o altri dischi.

È evidente infatti che alcune directory (come `/usr` ed `/opt`) possono essere mantenute su partizioni e filesystem diversi rispetto alla directory radice. Può risultare pertanto utile separare queste due directory che, contenendo file comuni di norma identici per le diverse installazioni,

possono essere montate in sola lettura e non inserite nei backup (in quanto è possibile sempre ripristinarle dall'installazione), o addirittura montate via rete e condivise fra più macchine.

La situazione è invece del tutto diversa per directory come `/home` e `/var`. Anche in questo caso può risultare utile separarle dalle altre directory, ma in questo caso è necessario l'accesso in scrittura e le informazioni variabili non saranno necessariamente condivisibili (ad esempio non lo sono `/var/run` e `/var/lock` che contengono informazioni sui processi locali). Inoltre essendo qui contenuti la gran parte dei dati del sistema (le altre directory sono solo `/root` per i file personali dell'amministratore e `/etc` per le configurazioni) queste dovranno essere sottoposte a regolare backup.

Si tenga inoltre presente che alcune di queste directory (ad esempio `/proc`) devono essere lasciate vuote sul disco; esse infatti servono solo come riferimento per montare i relativi filesystem virtuali. Non ha quindi alcun senso effettuare backup del contenuto di queste directory in quanto esse presentano solo una interfaccia di accesso (che permette però l'uso dei normali comandi per i file) a variabili interne del kernel create dinamicamente.

1.3 L'architettura dei processi

In questa sezione prenderemo in esame l'architettura della gestione dei processi, che costituiscono le entità fondamentali con cui il kernel permette l'esecuzione dei vari programmi. Vedremo come i processi sono organizzati in forma gerarchica, quali sono caratteristiche e proprietà che ciascuno di essi porta con sé, e come vengono gestiti all'interno del sistema.

1.3.1 Le caratteristiche dell'architettura dei processi

Come accennato in sez. 1.1.2 una delle caratteristiche principali dell'architettura dei processi in un sistema unix-like è che qualunque processo può creare nuovi processi; in questo contesto il processo originale viene chiamato *padre*, mentre i processi da lui creati vengono detti *figli*. La caratteristica distintiva è che tutti i processi presenti nel sistema possono essere creati solo in questo modo, e pertanto tutti i processi avranno un padre; l'unica eccezione è quella di `init`, il processo iniziale che, venendo lanciato direttamente dal kernel all'avvio, non è figlio di nessun altro processo.

Questa caratteristica permette di classificare i processi in una gerarchia ad albero basata sulla relazione padre-figlio; in questa gerarchia `init` viene a ricoprire nel sistema un ruolo speciale, come radice dell'albero. Questa classificazione può essere stampata con il comando `pstree` che evidenzia in maniera grafica l'*albero genealogico* dei processi presenti nel sistema, con un risultato del tipo:

```
piccardi@anarres:~$ pstree -A
init-+-acpid
     |-atd
     |-cron
     |-dhclient
     |-exim4
     |-5*[getty]
     |-login---bash
     |-portmap
     |-rpc.statd
```

```
|-rsyslogd---3*[rsyslogd]
|-sshd---sshd---sshd---bash---pstree
`-udevd---2*[udevd]
```

dove si può notare che, come dicevamo all'inizio, alla radice dell'albero c'è `init`.

Si tenga presenta che questa architettura, in cui qualunque processo può creare degli altri processi, è molto diversa da quella di altri sistemi operativi in cui spesso l'operazione di lanciare un nuovo processo è privilegiata e non può essere eseguita da un programma qualsiasi.

Nella creazione di un processo figlio questo eredita dal padre tutta una serie di proprietà (vedremo alcune di esse in sez. 1.3.2) e risorse,[30] ed in particolare una delle caratteristiche che contraddistingue l'architettura dei processi in un sistema unix-like è che tutti i file aperti nel processo padre resteranno tali ed immediatamente disponibili anche per il figlio;[31] vedremo in sez. 2.1.4 come questo sia fondamentale per una delle funzionalità più importanti dell'interfaccia a riga di comando.

Una seconda differenza rispetto ad altri sistemi multiutente come il VMS o Windows, è che in Linux la creazione di un processo e l'esecuzione di un programma sono due operazioni separate, gestite da due *system call* diverse,[32] la prima delle quali crea un nuovo processo, identico al padre, il quale normalmente va ad usare la seconda per eseguire un altro programma.[33]

Questa caratteristica permette anche, con una modalità di funzionamento comune per i server di rete, di scrivere un programma unico che esegue più processi, con una struttura in cui il padre esegue la parte che si occupa di ricevere le richieste, e per ciascuna di esse fa eseguire, ad un figlio creato appositamente, le operazioni necessarie a fornire le relative risposte.

1.3.2 Le proprietà dei processi

Il comando che permette di ottenere la lista dei processi attivi nel sistema è `ps`, il cui nome deriva da ***p**rocess **s**tatus*. Questo è uno dei comandi fondamentali presenti fin dalle prime versioni di Unix, per questo si sono accavallate anche diverse sintassi per le opzioni, derivate dalle varie versioni del comando che sono state realizzate nel tempo. La versione GNU del comando supporta la maggior parte delle opzioni esistenti, in genere indicate da lettere singole: quelle derivate da SysV devono essere precedute da un "`-`", quelle derivate da BSD *non* devono essere precedute da un "`-`", mentre le estensioni specifiche GNU usano un "`--`" ed una forma estesa.

Si tenga presente che in molti casi viene usata la stessa lettera, con significati diversi, sia nel formato BSD che in quello SysV, nel caso si vogliano combinare le due opzioni di questi formati alternativi allora non vale la sintassi usuale (che tratteremo in dettaglio in sez. 2.1.3) di specificare più opzioni con lettere consecutive dopo il "`-`", e ciascuna andrà ripetuta a parte.

[30] la trattazione dettagliata del meccanismo e di quali proprietà e risorse vengono trasmesse da padre a figlio va al di là dello scopo di questo testo, gli interessati possono consultare il capitolo 3 di [GaPiL].

[31] di nuovo si sta facendo una descrizione estremamente vaga e semplificata, in realtà il meccanismo con cui un processo figlio "*vede*" i file aperti dal padre è piuttosto complesso, una trattazione dettagliata si può trovare in sez. 5.2.1 di [GaPiL].

[32] le due funzioni si chiamano rispettivamente `fork` ed `exec`, per i dettagli del meccanismo si può consultare la sezione 3.1 di [GaPiL].

[33] è questa la modalità con cui funziona l'interfaccia a riga di comando in cui un programma, la shell, crea dei processi figli per eseguire tramite essi gli altri programmi.

Il comando non richiede nessun argomento, e se eseguito su un terminale senza specificare nessuna opzione `ps` mostra l'elenco dei processi appartenenti all'utente che ha eseguito il comando attivi su quel terminale; avremo cioè qualcosa del tipo:

```
piccardi@hain:~/truedoc/corso$ ps
  PID TTY          TIME CMD
10511 pts/2    00:00:00 bash
22461 pts/2    00:00:22 emacs
22746 pts/2    00:00:03 evince
22971 pts/2    00:00:00 ps
```

che mostra, nell'ultima colonna marcata `CMD`, come sul terminale siano presenti la shell, il comando `ps` stesso, più l'editor ed il visualizzatore di PDF utilizzati per la realizzazione di questo testo. Tralasciamo per ora il significato delle altre colonne, ci torneremo fra poco.

Specificando l'opzione "`a`" verranno visualizzati anche i processi lanciati da altri utenti, purché facenti riferimento ad un terminale, mentre con l'opzione "`x`" si visualizzano tutti i processi non associati ad un terminale, infine l'opzione "`f`" permette di mostrare la gerarchia dei processi. Così si può ottenere un elenco completo dei processi con un comando del tipo:

```
piccardi@anarres:~$ ps axf

  PID TTY      STAT   TIME COMMAND
    2 ?        S      0:01 [kthreadd]
    3 ?        S      0:00  \_ [migration/0]
...
  508 ?        S      0:00  \_ [kjournald]
    1 ?        Ss     0:04 init [2]
  217 ?        S<s    0:00 udevd --daemon
  266 ?        S<     0:00  \_ udevd --daemon
  279 ?        S<     0:00  \_ udevd --daemon
  724 ?        Sl     0:00 /usr/sbin/rsyslogd -c4
  773 ?        Ss     0:00 /usr/sbin/acpid
  786 ?        Ss     0:00 /usr/sbin/atd
  813 ?        Ss     0:00 /usr/sbin/cron
 1069 ?        Ss     0:00 /usr/sbin/exim4 -bd -q30m
 1087 tty1     Ss     0:00 /bin/login --
 1093 tty1     S+     0:00  \_ -bash
 1088 tty2     Ss+    0:00 /sbin/getty 38400 tty2
 1089 tty3     Ss+    0:00 /sbin/getty 38400 tty3
 1090 tty4     Ss+    0:00 /sbin/getty 38400 tty4
 1091 tty5     Ss+    0:00 /sbin/getty 38400 tty5
 1092 tty6     Ss+    0:00 /sbin/getty 38400 tty6
 1148 ?        Ss     0:00 /usr/sbin/sshd
 1151 ?        Ss     0:00  \_ sshd: piccardi [priv]
 1153 ?        S      0:00      \_ sshd: piccardi@pts/0
 1154 pts/0    Ss     0:00          \_ -bash
 1181 pts/0    R+     0:00              \_ ps axf
```

dato che la combinazione delle opzioni "`a`" e "`x`", essendo queste complementari, seleziona tutti i processi attivi nel sistema. L'uso dell'opzione "`r`" permette invece di restringere la selezione ai soli processi in esecuzione effettiva (cioè nello stato "`R`", che spiegheremo fra poco).

Come già questi primi esempi ci mostrano, in realtà lo scopo di `ps` non è solo quello di fornire la lista dei processi in esecuzione, quanto quello di mostrarne in generale caratteristiche e proprietà; la descrizione del significato di quest'ultime andrà allora di pari passo con la spiegazione dell'output del comando.

Per ogni processo attivo infatti il kernel mantiene tutta una serie di dati che vengono usati per il controllo delle varie operazioni che esso può compiere, insieme ad una serie di informazioni relative alle risorse da esso utilizzate. Già in questi due primi esempi il comando ci mostra alcune informazioni di base sufficienti ad introdurre alcune delle proprietà essenziali dei processi.

La prima colonna dei precedenti esempi, marcata `PID`, mostra il cosiddetto *process ID* del processo. Questo è il numero che il kernel utilizza per identificare univocamente ciascun processo; questo numero viene assegnato alla creazione del processo, ed è unico fintanto che il processo resta attivo. È questo il numero che si deve usare tutte le volte che si vuole fare riferimento ad uno specifico processo.

La seconda colonna, marcata `TTY`, mostra il nome del cosiddetto "*terminale di controllo*" del processo. Il concetto di "*terminale di controllo*" è alquanto complesso, e ci torneremo in sez. 2.1.2, per quanto ci interessa è sufficiente dire che un processo "*interattivo*" è sempre associato ad un terminale di controllo che corrisponde appunto al terminale da cui il processo riceve i dati in ingresso (in genere dalla tastiera) e sul quale scrive il suo output (in genere lo schermo).

I processi non interattivi (i cosiddetti *demoni*) che vengono usati per compiere una serie di compiti di servizio, come spedire la posta, eseguire lavori periodici, servire pagine web, ecc. dovendo essere attivi anche quando non c'è nessun utente collegato, lavorano invece come suol dirsi "in *background*" per cui non esiste un terminale di controllo ad essi associato e la colonna `TTY` riporta come valore "`?`".

In realtà questa caratterizzazione dell'interattività viene a cadere qualora si utilizzi una interfaccia grafica, dato che questa non necessita dell'uso di un terminale per operare in modo interattivo, ma Unix è nato molto prima dell'introduzione delle interfacce grafiche, ed a quel tempo l'uso di un terminale costituiva l'unica modalità di accesso al sistema. Per questo motivo il kernel fornisce una serie di servizi collegati a questo speciale dispositivo, ed uno dei punti più oscuri dell'interfaccia a linea di comando di un sistema unix-like è proprio quello relativo alla gestione delle *sessioni di lavoro*, su cui torneremo anche in sez. 2.1.2.

Si tenga presente che dal punto di vista del kernel non esiste nessuna differenza fra processi interattivi e non, o meglio fra processi con un terminale di controllo o meno; esso si limita a mettere a disposizione alcune risorse che possono essere utilizzate per realizzare il controllo di sessione, con la presenza di alcuni identificatori aggiuntivi come il *session ID* ed il *process group ID*, delle relative funzioni di gestione, e di alcuni segnali associati all'uso del terminale.[34]

Il controllo di sessione viene poi realizzato completamente in *user space*, usando queste risorse secondo le modalità classiche ereditate dai primi Unix. Anche questi identificativi possono essere visualizzati tramite `ps`, usando l'opzione `-j`, che ne stamperà il relativo valore nelle colonne `SID` e `PGID`; vedremo un esempio di tutto questo in sez. 2.1.2 quando tratteremo con qualche dettaglio in più la gestione delle cosiddette *sessioni di lavoro*.

In tutti i precedenti esempi compare, nella colonna marcata `STAT`, un'altra informazione fondamentale: lo stato del processo. Il sistema infatti prevede cinque possibili stati diversi per i processi, i cui valori sono riportati in tab. 1.10; torneremo su questo a breve, in quanto ci permette di spiegare varie caratteristiche dell'architettura dei processi. Seguono infine la colonna `TIME` che indica il tempo di CPU usato finora dal processo e la colonna `COMMAND` che riporta la riga di comando usata per lanciare il programma.

[34] per i dettagli su questi identificativi si consulti sez. 8.1.3 di [GaPiL].

Per capire il significato del campo `STAT` occorrono alcune spiegazioni generali sull'architettura della gestione dei processi. Come sistema multitasking Linux è in grado di eseguire più processi contemporaneamente, in genere però un processo, pur essendo attivo, non è assolutamente detto che sia anche in esecuzione. Il caso più comune infatti è quello in cui il programma è in attesa di ricevere dati da una periferica; in tal caso il kernel pone il programma in stato di *sleep* e lo toglie dalla lista di quelli che hanno bisogno del processore (che sono identificati invece dallo stato *runnable*).

Stato	`STAT`	**Descrizione**
runnable	`R`	il processo è in esecuzione o è pronto ad essere eseguito (cioè è in attesa che gli venga assegnata la CPU).
sleep	`S`	il processo è in attesa di una risposta dal sistema, ma può essere interrotto da un segnale.
uninterruptible sleep	`D`	il processo è in attesa di una risposta dal sistema (in genere per I/O), e non può essere interrotto in nessuna circostanza.
stopped	`T`	il processo è stato fermato con un `SIGSTOP`, o è tracciato.
zombie	`Z`	il processo è terminato ma il suo stato di terminazione non è ancora stato letto dal padre.

Tabella 1.10: Elenco dei possibili stati di un processo in Linux, nella colonna `STAT` si è riportata la corrispondente lettera usata dal comando `ps` nell'omonimo campo.

Si noti allora come nell'elenco precedente tutti i processi, eccetto lo stesso comando `ps axf`, fossero in stato di *sleep*. In genere un processo entra in stato di *sleep* tutte le volte che si blocca nell'esecuzione di una *system call* che richiede una qualche forma di accesso non immediato a dei dati: un caso classico è la lettura dell'input da tastiera.

Come dettagliato in tab. 1.10 gli stati di *sleep* sono due, contraddistinti dalle lettere `S` e `D`. Il più comune è il primo; l'esecuzione della maggior parte delle *system call* infatti può essere interrotta da un segnale (i segnali saranno trattati in sez. 1.3.3), per cui se un processo si blocca nell'esecuzione di una di queste esso può essere comunque terminato, in alcuni casi (in genere questo avviene quando la *system call* sta gestendo la risposta ad un interrupt hardware) questo non è possibile ed allora si ha uno stato di *uninterruptible sleep*, in questo caso se il processo resta bloccato[35] nello stato `D` non può più essere terminato se non con il riavvio della macchina.[36] Un processo in stato `D` comunque non avrà nessun effetto né sugli altri processi né sul sistema, che continuerà a funzionare senza problemi a parte quello di non poter liberare le risorse occupate dal processo.

Lo stato di *stopped* è relativo ai processi che vengono tracciati,[37] o la cui esecuzione è stata fermata per una richiesta dell'utente (per fare questo, come vedremo in sez. 1.3.3, si ha a disposizione un segnale apposito), nel qual caso semplicemente il processo non viene eseguito (ma resta in memoria e può riprendere l'esecuzione in qualunque momento) fintanto che non lo si fa ripartire (anche per questo è previsto un altro segnale).

[35]in genere questo avviene in maniera permanente solo per un qualche errore nella gestione della periferica nel kernel, ad esempio se si monta un disco USB e poi si lo si estrae dal bus senza smontarlo e si ha la sventura di farlo in un momento poco opportuno, o se ci sono problemi hardware o con un filesystem di rete come NFS in caso di crollo della connessione.

[36]in realtà con i kernel più recenti è stata introdotta una variante dello stato *uninterruptible sleep*, denominata *killable*, che consente, per i filesystem di rete (in sostanza NFS), di terminare il processo, ma `ps` non la distingue.

[37]si dice che un processo è tracciato se su di esso si usa il comando `strace`, che consente di intercettare e stampare a video tutte le *system call*, vedi sez. 2.4.5.

Infine c'è lo stato di *zombie*, il cui significato può essere compreso solo ritornando con maggiori dettagli sulla relazione fra processo padre e gli eventuali figli. La relazione padre/figlio infatti è ben definita finché entrambi i processi sono attivi nel sistema, ma se uno dei due termina cosa accade? Alla terminazione di un processo il kernel provvede ad una serie di compiti di pulizia; tutti i file aperti dal processo vengono chiusi, la memoria utilizzata viene liberata e con essa tutte le risorse occupate dal processo. Resta il problema di come notificare l'avvenuta conclusione del processo, ad esempio se questa è stata regolare o è stata dovuta ad un qualche errore, e il problema è a chi fare questa notifica.

Dato che in un sistema Unix tutto viene fatto con i processi, la scelta è stata quella di assegnare al padre il compito di ricevere lo stato di uscita del figlio e controllare se tutto è andato bene o c'è stato qualche errore. Quando un processo termina al padre viene inviato un apposito segnale che lo avvisa del fatto, e lui deve invocare una apposita *system call* per ricevere lo stato di uscita. Questo ad esempio è il meccanismo con cui la shell riceve lo stato di uscita dei comandi che si sono eseguiti.

Se questo non viene fatto il processo figlio si conclude comunque regolarmente e tutte le risorse che occupava nel sistema vengono rilasciate, resta allocata soltanto una voce nella tabella dei processi che contiene le informazioni per riportare lo stato di uscita. Questo è quello che in gergo viene chiamato uno *zombie*, cioè un processo che non esiste più, perché è terminato, ma che mostra una voce con lo stato "`Z`" nella lista fornita da `ps` e che di nuovo non può essere terminato, per il semplice fatto che lo è già. È pertanto compito di chi scrive programmi che creano processi figli curarsi della ricezione del loro stato di uscita, ed infatti i programmi ben scritti non presentano mai questo problema.

Di per sé la presenza di uno *zombie* non è un grave problema, in quanto esso non occupa, a differenza di un processo bloccato in stato "`D`", nessuna risorsa nel sistema a parte la voce mantenuta nell'output di `ps`. Però la voce nella tabella dei processi resta occupata e pertanto se il numero degli *zombie* dovesse crescere in maniera eccessiva si può rischiare di saturare quest'ultima, rendendo di fatto inutilizzabile il sistema, dato che non sarà più possibile creare nuovi processi. Ma a differenza dei processi bloccati permanentemente in stato "`D`" per gli *zombie* è possibile risolvere il problema, e far sì che essi siano terminati regolarmente, senza dover riavviare il sistema.

Per capire come comportarsi con gli *zombie* si deve considerare un altro caso della gestione del funzionamento dei processi: quello in cui è il padre a terminare prima del figlio. Se accade questo chi è che riceverà lo stato di terminazione dei suoi figli? Dato che i processi sono eseguiti in maniera del tutto indipendente un caso come questo è assolutamente naturale, e in tal caso si dice che il figlio diventa "*orfano*". Per questo il kernel controlla, durante le operazioni di terminazione di un processo, se questo ha dei figli, e nel caso assegna a questi ultimi `init` come nuovo padre.[38] Si dice allora che `init` *adotta* i figli dei processi che terminano, e sarà lui che gestirà, alla terminazione di questi ultimi, la ricezione del loro stato di uscita.

Perciò, dato che `init` è scritto bene e sa gestire la ricezione dello stato di uscita, tutto quello che serve fare per eliminare gli *zombie* presenti nel sistema è di renderli orfani, terminando il processo che li ha generati.[39] In questo modo essi saranno adottati da `init` che si curerà

[38] si vedrà cioè, ad una successiva esecuzione di `ps` con opzioni che permettano di visualizzare il PID del padre, che questo è diventato 1.

[39] per poterlo fare però occorre avere un processo in grado di eseguire il comando di terminazione, cosa non facile da ottenere se si è già riempita la tabella dei processi.

immediatamente di riceverne lo stato di uscita liberando la voce che occupavano nella tabella dei processi.

Si noti nella lista precedente la presenza di alcuni processi con un nome del comando fra parentesi quadre e con un PID molto basso;[40] questi non sono processi in *user space* lanciati da `init` quanto dei processi interni al kernel da esso utilizzati per la gestione di alcuni compiti, come lo scaricamento su disco dei buffer dei file, la generazione di eventi dovuti all'inserimento di periferiche, ecc. Questa notazione sta ad indicare che questi processi non usano memoria in *user space*, il che permette di identificarli. Le altre lettere associate al campo `STAT` sono "`<`", "`N`" e "`L`" e indicano rispettivamente una priorità maggiore o minore di quella standard (torneremo sulla priorità in sez. 1.3.4) e la presenza di pagine di memoria bloccate.[41] A queste si aggiungono ulteriori informazioni indicate dalle lettere minuscole, per il significato delle quali si rimanda alla pagina di manuale di `ps`.

Le opzioni di visualizzazione di `ps` sono moltissime, e qui potremo prendere in esame solo le principali. Restando nell'ambito della sintassi BSD una delle più usate è "`u`" che stampa una lista con le informazioni più rilevanti riguardo l'utente che ha lanciato il programma. Altre opzioni sono "`v`" che stampa informazioni relative all'uso della memoria virtuale, e `s` che stampa informazioni sui segnali. Infine l'opzione "`o`" permette all'utente di specificare un suo formato, con l'uso di una serie di direttive `%X`, dove "`X`" indica quale proprietà del processo si vuole far comparire nella lista (secondo una tabella di valori riportata nella pagina di manuale).

Se invece si usa la sintassi SysV le opzioni più usate sono `-e`, che permette di selezionare tutti i processi presenti, e `-f` che permette di avere una lista con più informazioni; in tal caso infatti si avrà come uscita del comando:

```
piccardi@anarres:~$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 12:39 ?        00:00:04 init [2]
root         2     0  0 12:39 ?        00:00:00 [kthreadd]
root         3     2  0 12:39 ?        00:00:00 [migration/0]
...
root       217     1  0 12:39 ?        00:00:00 udevd --daemon
root       266   217  0 12:39 ?        00:00:00 udevd --daemon
root       279   217  0 12:39 ?        00:00:00 udevd --daemon
root       508     2  0 12:39 ?        00:00:00 [kjournald]
root       724     1  0 12:39 ?        00:00:00 /usr/sbin/rsyslogd -c4
root       773     1  0 12:39 ?        00:00:00 /usr/sbin/acpid
daemon     786     1  0 12:39 ?        00:00:00 /usr/sbin/atd
root       813     1  0 12:39 ?        00:00:00 /usr/sbin/cron
101       1069     1  0 12:39 ?        00:00:00 /usr/sbin/exim4 -bd -q30m
root      1087     1  0 12:39 tty1     00:00:00 /bin/login --
root      1088     1  0 12:39 tty2     00:00:00 /sbin/getty 38400 tty2
root      1089     1  0 12:39 tty3     00:00:00 /sbin/getty 38400 tty3
root      1090     1  0 12:39 tty4     00:00:00 /sbin/getty 38400 tty4
root      1091     1  0 12:39 tty5     00:00:00 /sbin/getty 38400 tty5
root      1092     1  0 12:39 tty6     00:00:00 /sbin/getty 38400 tty6
root      1093  1087  0 12:40 tty1     00:00:00 -bash
root      1148     1  0 12:40 ?        00:00:00 /usr/sbin/sshd
root      1151  1148  0 12:40 ?        00:00:00 sshd: piccardi [priv]
```

[40] prima del kernel 2.6 veniva riportata anche una lettera aggiuntiva "`W`" nella colonna `STAT`.

[41] quest'ultima è una caratteristica avanzata che va al di là degli scopi di questo testo, per una trattazione si può fare riferimento alla sezione 2.2.4 di [GaPiL].

```
piccardi  1153  1151  0 12:40 ?        00:00:00 sshd: piccardi@pts/0
piccardi  1154  1153  0 12:40 pts/0    00:00:00 -bash
piccardi  1180  1154  0 12:56 pts/0    00:00:00 ps -ef
```

Come si può vedere questa versione riporta una serie di dati in più. Anzitutto notiamo che la prima colonna, `UID`, riporta un nome utente. Ciascun processo infatti mantiene le informazioni riguardanti l'utente che ha lanciato il processo ed il gruppo cui questo appartiene, il kernel infatti identifica ogni utente e gruppo presente nel sistema con dei numeri interi, detti rispettivamente *user ID* (UID) e *group ID* (GID) che poi vengono abbinati a dei nomi simbolici come quelli riportati dal comando (tratteremo l'argomento in dettaglio in sez. 1.4.1).

Ogni processo in realtà mantiene diverse versioni di questi identificatori,[42] ma i più significativi sono i l'*effective user ID* ed l'*effective group ID*, che vengono utilizzati per il controllo di accesso (che tratteremo in sez. 1.4). I nomi corrispondenti a questi identificatori sono quelli che corrispondono alla colonna `UID` (e ad una eventuale colonna `GID`) del comando, ma è possibile anche richiedere il *real user ID* e il *real group ID*, che identificano l'utente che ha lanciato il processo,[43] che invece vengono identificati da sigle come `RUSER` e `RGROUP`, se indicati con il nome o con `RUID` e `RGID` se numerici.

La seconda colonna del nostro esempio riporta di nuovo il PID di ciascun processo, mentre la terza colonna, `PPID`, indica il *parent process ID*, ogni processo infatti mantiene il PID del padre, in modo da poter ricostruire agevolmente la genealogia dei processi. Questa ci permette anche, in caso di *zombie*, di identificare il processo responsabile della produzione degli stessi.

La quarta colonna, `C`, indica il valore medio, espresso come intero, della percentuale di utilizzo della CPU su tutta la vita del processo. La quinta colonna, `STIME`, indica invece il momento in cui il comando è stato lanciato. Le altre colonne sono analoghe a quelle già viste in precedenza per la sintassi BSD. Un elenco delle principali informazioni relative ai processi che vengono riportate da `ps` sono elencate in tab. 1.11, indicate sempre in base al nome usato come intestazione della colonna che ne visualizza il valore. Per l'elenco completo di opzioni, informazioni e dati visualizzabili si può fare riferimento alla pagina di manuale del comando, accessibile con `man ps`.

Infine ci sono alcune proprietà dei processi, non immediatamente visualizzabili con `ps`,[44] che comunque sono importanti e vanno menzionate. Come accennato in sez. 1.2.2, ogni processo ha una directory di lavoro rispetto alla quale risolve i *pathname* relativi; anche questa è una caratteristica del processo, che viene ereditata nella creazione di un processo figlio.

Benché questo possa sembrare strano a prima vista, lo stesso vale per la *directory radice*, ogni processo cioè porta con sé, oltre alla directory di lavoro corrente, anche l'indicazione della directory che considera come radice, rispetto alla quale risolve i *pathname* assoluti. Questa directory può essere cambiata con una apposita *system call* (chiamata `chroot`) ottenendo così che il processo venga in un certo senso *ristretto* all'interno di una parte dell'albero dei file (torneremo su questo in sez. 1.4.4).

---

[42] Linux usa ben quattro gruppi di identificatori diversi per gestire il controllo di accesso, secondo uno schema che va al di là di quanto è possibile spiegare qui, gli interessati possono trovare una trattazione dell'argomento nella sezione 3.2 di [GaPiL].

[43] di norma questi coincidono con gli identificatori del gruppo *effective*, ma vedremo in sez. 1.4.2 che esistono casi in cui questo non avviene.

[44] si possono comunque ricercare tutte le informazioni relative ai processi nel filesystem `/proc`, che è nato proprio per fornire una interfaccia con kernel da cui ottenerle (tanto che se non è montato `ps` non funzionerà); in particolare sotto `/proc` si troverà per ciascun processo una directory con nome uguale al rispettivo PID, il cui contenuto è costituito una serie di file e directory (virtuali) che permettono di accedere alle proprietà dello stesso.

Proprietà	Descrizione
`PID`	PID, *process ID* del processo.
`PPID`	PPID, *process ID* del padre del processo.
`UID`	*effective user ID* del processo.
`GID`	*effective group ID* del processo.
`CMD`	linea di comando con cui è stato lanciato il processo.
`STAT`	stato del processo.
`NI`	valore di *nice* (vedi sez. 1.3.4) del processo.
`TTY`	terminale di controllo del processo.
`SID`	SID, *session ID* (vedi sez. 2.1.2) del processo.
`PGID`	PGID, *process group ID* (vedi sez. 2.1.2) del processo.
`%CPU`	percentuale del tempo di CPU usato rispetto al tempo reale di esecuzione.
`%MEM`	percentuale della memoria fisica utilizzata dal processo.
`C`	percentuale di CPU sulla media della vita del processo.
`START`	orario dell'avvio del processo.
`TIME`	tempo totale di CPU utilizzato dall'avvio del processo.
`USER`	*effective user ID* del processo.
`RUSER`	*real user ID* del processo.
`GROUP`	*effective group ID* del processo.
`RGROUP`	*real group ID* del processo.
`COMMAND`	riga di comando con la quale si è avviato il processo.

Tabella 1.11: Le principali proprietà dei processi ed il nome della relativa colonna nella visualizzazione effettuata da `ps`.

Come si può notare `ps` si limita a stampare la lista dei processi attivi al momento della sua esecuzione. Se si vuole tenere sotto controllo l'attività del sistema non è pratico ripetere in continuazione l'esecuzione di `ps`. Per questo ci viene in aiuto il comando `top`, che stampa una lista di processi, aggiornandola automaticamente in maniera periodica.

Il comando opera normalmente in maniera interattiva, a meno di non averlo lanciato con l'opzione `-b`, che lo esegue in modalità batch, consentendo la redirezione dell'output; in tal caso di solito si usa anche l'opzione `-n` per specificare il numero di iterazioni volute. Il comando ristampa la lista ogni secondo, a meno di non aver impostato un intervallo diverso con l'opzione `-d`, che richiede un parametro nella forma *`ss.dd`* dove *`ss`* sono i secondi e *`dd`* i decimi di secondo. Infine l'opzione `-p` permette di osservare una lista di processi scelta dall'utente, che deve specificarli tramite una lista di PID. Per la lista completa delle opzioni si faccia riferimento al solito alla pagina di manuale, accessibile con `man top`.

Quando opera in modalità interattiva il comando permette di inviare dei comandi da tastiera. I più rilevanti sono "`h`" e "`?`" che mostrano un help in linea dei comandi disponibili, e "`q`" che termina il programma. Un esempio di output del comando è riportato in fig. 1.4.

In testa vengono sempre stampate cinque righe di informazioni riassuntive sul sistema: nella prima riga viene riportato l'orario, *uptime*,[45] il numero di utenti ed il carico medio della macchina; nella seconda riga le statistiche sul totale dei processi; nella terza le statistiche di utilizzo della CPU, nelle ultime due le statistiche di uso della memoria fisica e della *swap* (su cui torneremo in sez. 5.1.7).

A queste informazioni generiche seguono, dopo una riga lasciata vuota che serve per gestire l'input in interattivo, le informazioni sui processi, ordinati per uso decrescente della CPU (ven-

[45] cioè il tempo trascorso da quando il sistema è stato avviato.

```
piccardi@anarres: ~
File  Edit  View  Search  Terminal  Help
top - 15:14:21 up  2:15,  2 users,  load average: 0.00, 0.00, 0.00
Tasks:  59 total,   1 running,  58 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.3%sy,  0.0%ni, 46.4%id,  0.0%wa,  0.0%hi, 53.3%si,  0.0%st
Mem:    514684k total,    44836k used,   469848k free,     3072k buffers
Swap:   314360k total,        0k used,   314360k free,    22436k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1324 piccardi  20   0  2336 1128  904 R  1.0  0.2   0:00.14 top
 1279 piccardi  20   0  8264 1428  896 S  0.3  0.3   0:00.57 sshd
    1 root      20   0  2036  716  620 S  0.0  0.1   0:05.19 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.01 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:00.12 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    6 root      20   0     0    0    0 S  0.0  0.0   0:04.54 events/0
    7 root      20   0     0    0    0 S  0.0  0.0   0:00.00 cpuset
    8 root      20   0     0    0    0 S  0.0  0.0   0:00.00 khelper
    9 root      20   0     0    0    0 S  0.0  0.0   0:00.00 netns
   10 root      20   0     0    0    0 S  0.0  0.0   0:00.00 async/mgr
   11 root      20   0     0    0    0 S  0.0  0.0   0:00.00 pm
   12 root      20   0     0    0    0 S  0.0  0.0   0:00.24 sync_supers
   13 root      20   0     0    0    0 S  0.0  0.0   0:00.27 bdi-default
   14 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kintegrityd/0
   15 root      20   0     0    0    0 S  0.0  0.0   0:00.02 kblockd/0
   16 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kacpid
```

Figura 1.4: Schermata del comando `top`.

gono cioè mostrati i più attivi), evidenziando (con la stampa in grassetto) quelli trovati in stato *runnable*. Le informazioni riportate di default sono il PID del processo (colonna omonima), l'utente cui esso appartiene (colonna `USER`); la priorità ed il valore di *nice* (torneremo su questi in sez. 1.3.4) rispettivamente nelle colonne `PR` e `NI`.

Seguono i dati dell'uso della memoria nelle colonne `VIRT`, `RES` e `SHR`; per capire queste quantità occorre dare qualche dettaglio in più sul sistema della *memoria virtuale* cui abbiamo già accennato in sez. 1.1.1; in quella occasione abbiamo detto come sia compito del kernel mappare lo spazio (virtuale) degli indirizzi di memoria di un processo nella memoria fisica effettivamente disponibile. La memoria usata da un processo è sostanzialmente suddivisa in due parti,[46] il codice del programma e i dati usati dallo stesso.

Il programma per essere eseguito dovrà avere il codice ed i relativi dati nella memoria fisica, e la quantità di memoria attualmente usata a questo scopo è quella che è indicata dalla colonna `RES` e viene detta *residente*, che sarà la somma della parte usata per i dati (indicata da `DATA`) e della parte usata per il codice (indicata da `CODE`). Una parte di quest'ultima (ad esempio il codice delle librerie) sarà condivisa con altri processi, e questa viene indicata dalla colonna `SHR`.

[46] si sta usando una descrizione brutalmente semplificata, per una trattazione dettagliata dell'argomento si può fare riferimento alla sezione 2.2 di [GaPiL].

Ci sarà però anche una parte di memoria che al momento non è in uso, e che, per liberare della memoria fisica a favore di altri programmi che devono essere eseguiti, è stata temporaneamente *parcheggiata* su una area di disco a questo dedicata (detta *swap*, vedi sez. 5.1.7) da cui può essere ripresa in caso di necessità. Allora il totale della memoria vista dal programma, che a questo punto è *virtuale*, in quanto non corrisponde a della RAM direttamente accessibile, è quello espresso dalla colonna `VIRT`, che sarà la somma sia della parte *residente* (quella di `RES`) che di quella che è stata parcheggiata nell'area di *swap*.

Come altre informazioni presenti nella stampa di default il comando riporta lo stato del processo (colonna `S`), le percentuali di utilizzo di CPU e memoria (colonne `%CPU` e `%MEM`), il tempo di CPU utilizzato dall'avvio del programma (colonna `TIME+`) ed il comando usato per lanciarlo (colonna `COMMAND`), il cui significato è analogo a quello già visto per `ps`.

Proprietà	**Descrizione**
`%CPU`	percentuale di uso della CPU dall'ultimo aggiornamento dello schermo.
`%MEM`	percentuale di uso della memoria da parte del processo.
`CODE`	ammontare della memoria fisica utilizzata dal processo per il suo codice eseguibile.
`DATA`	ammontare della memoria fisica utilizzata dal processo per i suoi dati.
`RES`	ammontare della memoria fisica usata dal processo (uguale a `CODE` + `DATA`).
`S`	stato del processo (analogo allo `STAT` di `ps`).
`SHR`	ammontare della *memoria condivisa*, rappresenta la memoria potenzialmente condivisibile con altri processi.
`SWAP`	ammontare della *memoria virtuale* di un processo presente nella *swap*.
`TIME+`	tempo di CPU utilizzato dall'avvio, analogo al `TIME` di `ps`, ma con granularità fino al centesimo di secondo.
`VIRT`	ammontare totale della *memoria virtuale* usata dal processo include tutto il codice, i dati e le librerie condivise più le pagine che sono state messe su *swap* (uguale a `SWAP` + `RES`).

Tabella 1.12: Proprietà dei processi e nome della relativa colonna nella visualizzazione effettuata da `top`.

In generale le informazioni riportate nelle colonne stampate da `top` sono simili a quelle di `ps`, anche se in alcuni casi esse vengono indicate diversamente. Le principali proprietà mostrate da `top` sono riportate in tab. 1.12, per quelle comuni non presenti in detta tabella si faccia riferimento alle omonime di `ps` riportate in tab. 1.11.

Come accennato se `top` viene usato in modalità interattiva diventa possibile inviare una serie di comandi da tastiera, ad esempio con "`k`" si può inviare un segnale (di default è `SIGTERM`, vedi sez. 1.3.3) mentre con "`r`" si può cambiare la priorità (vedi sez. 1.3.4) di un processo. Con il comando "`u`" si possono selezionare i processi di un utente, con "`c`", si può alternare fra la stampa del nome comando e della riga completa, con "`d`" cambiare il periodo di aggiornamento dei risultati. L'elenco completo, oltre ad essere riportato nella pagina di manuale, può essere stampato a video con "`h`".

Si noti che nelle righe iniziali mostrate in precedenza `top` riporta anche delle statistiche complessive sull'uso della memoria, queste possono essere ottenute separatamente tramite il comando `free`, che mostra un riassunto generale dell'impiego della memoria all'interno del sistema, con un qualcosa del tipo:

```
piccardi@anarres:~$ free
             total       used       free     shared    buffers     cached
```

```
Mem:        775444     759364      16080          0     213276     316908
-/+ buffers/cache:     229180     546264
Swap:       498004      34708     463296
```

La prima riga riporta l'uso della memoria fisica, mentre l'ultimo quello della *swap* (se questa è attiva, torneremo sull'argomento in sez. 5.1.7). La riga centrale ci dice quanto della memoria è occupata o libera dal punto di vista delle applicazioni, per cui al valore alle colonne `used` e `free` riportato nella prima riga viene rispettivamente sottratta e aggiunta la somma delle colonne `buffers` e `cached` della stessa, dato che esse indicano i dati temporanei che possono essere scartati qualora sia necessaria altra memoria per le applicazioni. In questo modo si ottiene la memoria usata direttamente dalle applicazioni e il totale di quella che potrebbe essere loro messa a disposizione.

Si noti che in genere la RAM libera è sempre molto poca; questo è corretto, in quanto non ha senso lasciare inutilizzata la RAM, per cui viene impiegata dai buffer del kernel (per gestire più efficacemente il trasferimento dei dati verso i dispositivi) e per mantenere dati temporanei, come riportato nelle colonne `buffers` e `cached`.[47]

Il comando `free` non ha argomenti e prende come opzioni `-b`, `-k` e `-m` per stampare i dati di utilizzo rispettivamente in byte, kilobyte (il default) e megabyte. Inoltre lo si può mantenere attivo in modo da osservare continuamente lo stato della memoria usando l'opzione `-s`, seguita dal numero di secondi che passano fra un controllo e l'altro. L'elenco completo delle opzioni ed i relativi dettagli sono al solito riportati nella pagina di manuale, accessibile con `man free`.

Infine qualora si voglia ricercare il valore del PID di un certo programma, invece di andare a cercare nell'output di `ps` si può utilizzare `pidof`. Il comando prende come argomento una lista di nomi dei programmi di cui si vuole conoscere il PID ed esegue automaticamente una ricerca nella tabella dei processi, stampando la lista dei PID di tutti i processi che corrispondono. Il comando prende l'opzione `-s`, che richiede, in caso di più processi che corrispondono allo stesso programma, di stampare un solo PID, mentre gli si può dire di omettere un PID passandolo come parametro per l'opzione `-o`. Per i dettagli del funzionamento e l'elenco completo delle opzioni si può consultare la pagina di manuale accessibile con `man pidof`.

Il comando viene usato in genere negli script di shell (vedi sez. 2.1.5) per controllare se un certo programma è in esecuzione; infatti se nella ricerca non viene trovato nessun processo il comando non stampa niente ma ritorna uno stato di uscita indicante un errore, che può essere usato per prendere delle decisioni. Si tenga presente che la ricerca viene eseguita sul nome del programma, se esistono diversi programmi con lo stesso nome per avere la certezza di scegliere quello voluto è necessario indicare un *pathname* assoluto.

1.3.3 I segnali

Benché i processi siano di norma entità separate e completamente indipendenti fra di loro, esistono molti casi in cui è necessaria una qualche forma di comunicazione. La forma più elementare di comunicazione fra processi è costituita dai segnali, che sono usati anche direttamente dal kernel per comunicare ai processi una serie di eventi o errori (come l'uso inappropriato della memoria o una eccezione aritmetica).

[47] la colonna `shared` veniva usata per indicare la memoria condivisa fra più processi, adesso è obsoleta e deve essere ignorata.

Come dice la parola un segnale è una specie di avviso che viene inviato ad un processo; e non contiene nessuna informazione oltre al fatto di essere stato inviato. In genere i segnali vengono utilizzati per notificare ai processi una serie di eventi (abbiamo accennato in sez. 1.3.2 che uno di essi viene utilizzato per notificare la terminazione di un processo figlio), e possono essere anche inviati a mano attraverso l'uso del comando `kill`.[48] Ciascun segnale è identificato da un numero e da un nome simbolico; la lista dei segnali disponibili può essere ottenuta semplicemente con:

```
piccardi@anarres:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR      31) SIGSYS      32) SIGRTMIN    33) SIGRTMIN+1
34) SIGRTMIN+2  35) SIGRTMIN+3  36) SIGRTMIN+4  37) SIGRTMIN+5
38) SIGRTMIN+6  39) SIGRTMIN+7  40) SIGRTMIN+8  41) SIGRTMIN+9
42) SIGRTMIN+10 43) SIGRTMIN+11 44) SIGRTMIN+12 45) SIGRTMIN+13
46) SIGRTMIN+14 47) SIGRTMIN+15 48) SIGRTMAX-15 49) SIGRTMAX-14
50) SIGRTMAX-13 51) SIGRTMAX-12 52) SIGRTMAX-11 53) SIGRTMAX-10
54) SIGRTMAX-9  55) SIGRTMAX-8  56) SIGRTMAX-7  57) SIGRTMAX-6
58) SIGRTMAX-5  59) SIGRTMAX-4  60) SIGRTMAX-3  61) SIGRTMAX-2
62) SIGRTMAX-1  63) SIGRTMAX
```

I segnali effettivamente usati dal sistema sono i primi 31, gli altri sono chiamati *real time signal* ed hanno un uso specialistico che va al di là di quello che possiamo affrontare qui.[49] In generale ciascuno di essi ha un compito o un significato specifico, alcuni di essi vengono generati automaticamente in caso di errori del programma. Ad esempio il segnale `SIGSEGV` viene inviato dal kernel per segnalare ad un processo una *Segment Violation*, cioè un accesso illegale alla memoria;[50] altri vengono inviati direttamente dall'utente, come `SIGKILL` che causa l'immediata terminazione di un processo.

Gran parte dei segnali (tutti eccetto `SIGKILL` e `SIGSTOP`, che hanno un comportamento speciale) possono essere *intercettati* dal processo, che può eseguire una opportuna funzione al loro arrivo. Ad esempio il segnale `SIGTERM` (che è quello che il comando `kill` invia di default) serve per richiedere la terminazione immediata di un processo, ma il segnale può essere intercettato per eseguire delle operazioni di pulizia, come cancellare dei file temporanei prima dell'uscita.

Se un segnale non viene intercettato viene eseguita un'azione di default che è specifica di ciascuno di essi, ma che nella maggior parte dei casi consiste nella terminazione immediata del processo. Questa può però avvenire in due modi, o con una semplice uscita o con l'uscita eseguita insieme alla creazione, nella directory di lavoro del processo, di un file `core` che contiene una copia del suo spazio di memoria (e per questo viene chiamato *core dump*) che può essere usato per ulteriori analisi per vedere in che punto è avvenuta l'interruzione. Infine alcuni segnali (come

[48] con la `bash` shell il comando è disponibile anche come comando interno, nel qual caso può avere un comportamento leggermente diverso per alcune delle opzioni rispetto a quanto riportato qui.

[49] per una spiegazione più dettagliata al riguardo si può fare riferimento alla sezione 7.5.1 di [GaPiL].

[50] questo è un altro dei vantaggi della memoria virtuale, se per un errore di programmazione un processo cerca di scrivere su una parte dello spazio degli indirizzi che non corrisponde a nessuna locazione di memoria associata al processo stesso, il sistema se ne accorge ed invia questo segnale che ne causa la terminazione; si impedisce così che un accesso sbagliato possa andare a sovrascrivere la memoria di altri processi.

SIGCHLD, che è quello che viene usato per notificare al padre la terminazione di un figlio) di default vengono ignorati,[51] il programma cioè continua ad essere eseguito senza nessuna conseguenza.

Tipici segnali che causano una semplice uscita sono SIGTERM, il default del comando kill, e SIGINT che è associato al carattere di interruzione dato dalla tastiera (vedi sez. 2.1.2), un segnale che invece produce un *core dump* è SIGQUIT, così come tutti i segnali relativi ad errori di programmazione come SIGSEGV, SIGFPE, ecc. Segnali come SIGSTOP e SIGTSTP invece hanno come effetto quello di fermare l'esecuzione di un programma, mandandolo in stato T. Un elenco dettagliato dei segnali, del loro significato e delle azioni di default si può trovare nella sezione 9.2 di [GaPiL].

In generale il comando kill permette di inviare un segnale ad un processo qualunque, specificando come parametro il PID di quest'ultimo. Come accennato il segnale inviato di default è SIGTERM, ma si può inviare qualunque altro segnale specificandone numero o nome preceduto da un "-" come opzione; ad esempio:

```
kill -9 1029
kill -SIGKILL 1029
kill -KILL 1029
kill -s SIGKILL 1029
```

sono modalità equivalenti di inviare il segnale SIGKILL al processo con PID 1029. Oltre a -s e -l il comando kill accetta le opzioni -L (sinonimo di -l) e -V che ne stampa la versione. Per una descrizione accurata delle opzioni si faccia al solito riferimento alla pagina di manuale accessibile con man kill.

Nel caso specifico, dato che SIGKILL non è intercettabile e la sua azione di default è la terminazione del processo, l'effetto di questo comando è di terminare senza possibilità di scampo il processo in questione. Infatti SIGTERM può venire intercettato e può risultare inefficace qualora il processo rimanga bloccato nell'esecuzione della funzione di gestione; ma siccome non è possibile intercettare SIGKILL si ha a disposizione un mezzo infallibile[52] per terminare un processo "*impazzito*", e questo senza alcuna conseguenza per gli altri processi o per il sistema.

I segnali sono un meccanismo di comunicazione elementare fra processi, è cioè possibile utilizzarli per dare delle istruzioni (molto semplici, come quella di terminare l'esecuzione) ad un processo. Uno dei segnali più usati come argomento del comando kill è ad esempio SIGHUP, che viene spesso utilizzato per dire ai *demoni* di sistema di rileggere il proprio file di configurazione. Inoltre alcuni segnali, come SIGTSTP, SIGINT e lo stesso SIGHUP, sono associati al controllo del terminale e vengono inviati attraverso opportune combinazioni di tasti o in occasione di eventi particolari; torneremo su questo in sez. 2.1.2 quando affronteremo le questioni relative al controllo di sessione.

Infine invece che ad un singolo processo si può inviare un segnale ad un intero *process group* (sui *process group* torneremo più avanti in sez. 2.1.2) usando come argomento il negativo del relativo *process group ID*; il valore -1 inoltre ha il significato speciale di indicare tutti i processi correnti, eccetto init ed il processo che esegue il comando.

---

[51] nel caso di SIGCHLD non è proprio il caso di farlo, altrimenti, come spiegato in sez. 1.3.2, ci si ritroverà con degli *zombie*.

[52] in realtà quasi infallibile, infatti come accennato in sez. 1.3.2, inviare segnali ad un processo in stato "D" non ha effetto, perché non può riceverli fintanto che resta in quello stato, ed è inutile inviarli ad un processo in stato "Z" perché in realtà esso non esiste più.

Un comando alternativo a `kill`, e più facile da usare, è `killall` che invece di richiedere un numero di PID funziona indicando il nome del programma, ed invia il segnale (specificato con la stessa sintassi di `kill`) a tutti i processi attivi con quel nome. Le opzioni principali disponibili sono riportate in tab. 1.13. Al solito la documentazione completa è nella pagina di manuale, accessibile con `man killall`.

Opzione	**Significato**
`-g`	invia il segnale al *process group* del processo.
`-e`	richiede una corrispondenza esatta anche per nomi molto lunghi (il comando controlla solo i primi 15 caratteri).
`-i`	chiede una conferma interattiva prima di inviare il segnale.
`-l`	stampa la lista dei nomi dei segnali.
`-u` *`user`*	manda il segnale ai processi dell'utente *`user`*.
`-w`	attende la terminazione di tutti i processi cui ha inviato il segnale.

Tabella 1.13: Principali opzioni del comando `killall`.

Altri due comandi ancora più sofisticati per la gestione dei segnali e l'identificazione dei processi, sono `pgrep` e `pkill` (in realtà il secondo è una modalità alternativa di chiamare il primo). Con `pgrep` infatti è possibile selezionare gruppi di processi, cui eventualmente inviare, se invocato come `pkill`, un segnale che di default è `SIGTERM` ma che può essere uno qualunque, da indicare con la stessa sintassi di `kill`. Entrambi i comandi prendono come argomento una espressione regolare (vedi sez. 2.2.5) che viene usata per selezionare i processi usando il relativo nome. La ricerca può essere ristretta ad una corrispondenza esatta usando l'opzione `-x` o estesa all'intera riga di comando con cui è stato lanciato il processo con `-f`. Si può anche invertire la selezione con `-v`, prendendo tutti i processi che non corrispondono.

Opzione	**Significato**
`-f`	estende la selezione all'intera riga di comando.
`-G` *`gid,name,`*	restringe la selezione ai processi con GID nella lista indicata.
`-n`	seleziona il processo più recente.
`-o`	seleziona il processo meno recente.
`-P` *`ppid,`*	restringe la selezione ai processi con padre nella lista indicata.
`-s` *`sid,`*	restringe la selezione ai processi con *session Id* nella lista indicata.
`-t` *`tty,`*	restringe la selezione ai processi con terminale di controllo nella lista indicata.
`-u` *`uid,name,`*	restringe la selezione ai processi con *effective user ID* nella lista indicata.
`-U` *`uid,name,`*	restringe la selezione ai processi con *real user ID* nella lista indicata.
`-v`	inverte la selezione.
`-x`	richiede una corrispondenza esatta al nome del comando.
`-d` *`delim`*	stampa l'elenco usando il carattere `delim` come separatore al posto dell'a capo.

Tabella 1.14: Principali opzioni dei comandi `pgrep` e `pkill`.

Oltre che al nome è possibile restringere ulteriormente la selezione dei processi in base a tutta una serie di altre loro caratteristiche, ad esempio si possono selezionare per utenti passando una

lista separata da virgole, sia di username che di UID, all'opzione `-u`, mentre per i gruppi si può usare in modo analogo l'opzione `-G`, si possono poi selezionare i processi per terminale di controllo, *session ID*, *process group*, PPID, ecc. Infine possono essere molto comode le opzioni `-n` e `-o` per selezionare rispettivamente il più ed il meno recente (in termini di tempo di avvio). Un elenco delle opzioni più importanti è riportato in tab. 1.14, nella cui seconda parte si sono inserite le opzioni specifiche del solo `pgrep`.

1.3.4 La gestione delle priorità

Abbiamo visto in sez. 1.3.2 che una delle proprietà dei processi che si può modificare con `top` è la priorità. In realtà, come accennato in tale occasione, quello che di norma si può modificare non è tanto la priorità di un processo, quanto il suo valore di *nice*.

La gestione delle priorità in un sistema unix-like infatti è abbastanza complessa dato che esistono due tipi di priorità: statiche e dinamiche. Di norma le priorità statiche vengono utilizzate solo per i cosiddetti processi *real-time*,[53] i processi ordinari (tutti, l'uso di processi *real-time* è riservato a pochi usi specialistici) hanno priorità statica nulla e il loro ordine di esecuzione viene stabilito solo in base a delle priorità dinamiche.

Una priorità statica più alta comporta che un processo verrà sempre eseguito prima di ogni altro processo a priorità più bassa. Il che vuol dire che se si lancia un processo a priorità statica alta che non fa I/O non si potrà fare più nulla nel sistema fintanto che questo non si sarà concluso (e se c'è un errore e il programma si blocca in un ciclo si dovrà necessariamente riavviare la macchina, non avendo modo di eseguire nient'altro). La cosa non vale quando il processo deve fare I/O perché in tal caso anche il processo *real-time* viene messo in stato di *sleep*, ed in quel momento altri processi possono essere eseguiti, lasciando così la possibilità di interromperlo. Se più processi hanno la stessa priorità statica l'ordine di esecuzione dipende dalla politica di *scheduling* scelta, che può essere di tipo *Round Robin* (RR), in cui i processi girano a turno per un tempo fisso, e *First In First Out* (FIFO), in cui vengono eseguiti nella sequenza in cui sono stati lanciati.

Nell'uso normale comunque non si ha bisogno di usare le priorità statiche (che come accennato sono anche molto rischiose in caso di errori di programmazione), e ci si affida al normale procedimento di *scheduling*, basato su un sistema di priorità dinamiche che permette di ottenere quella che usualmente viene chiamata la "*fairness*" nella distribuzione del tempo di CPU. Tralasciando i dettagli[54] possiamo dire che le priorità dinamiche sono caratterizzate da un valore iniziale, che è quello che poi si chiama *nice*, che di default è nullo; più un processo viene eseguito, più il valore di priorità dinamica aumenta, e lo *scheduler* mette sempre in esecuzione, fra tutti quelli in stato *runnable*, il processo che ha una priorità dinamica più bassa. Questo fa sì che anche i processi che partono da un valore di *nice* più alto (che viene chiamato così appunto perché i processi che lo usano sono più "*gentili*" nei confronti degli altri) ottengano alla fine una possibilità di essere eseguiti, secondo quello che appunto è il meccanismo chiamato *fairness*.

Il comando che permette di modificare il valore di *nice* di un processo è appunto `nice`, che deve essere usato per lanciare un altro comando, con una sintassi in cui si fa seguire a `nice` la

[53] questo nome è in realtà fuorviante, Linux, almeno nella sua versione standard, non è un sistema operativo *real-time*. Se si ha necessità di usare un sistema effettivamente *real-time* occorre usare una versione del kernel opportunamente modificata, ad esempio Xenomai (`http://www.xenomai.org`).

[54] una trattazione più dettagliata si può trovare in sez. 3.3 di [GaPiL].

linea di comando di cui si vuole cambiare la priorità di esecuzione. L'opzione principale è `-n` (da specificare prima della riga di comando che si vuole eseguire) che permette di specificare un valore di *nice* da applicare al programma. Se non si specifica nulla viene applicato un valore di default pari a 10. Si ricordi che valori positivi corrispondono ad una diminuzione della priorità. Solo l'amministratore può anche usare valori negativi, aumentando così la priorità di un processo, i valori possibili sono comunque limitati all'intervallo fra 19 e -20.

Il comando `nice` può essere usato solo quando si avvia un programma, se si vuole cambiare la priorità di un programma già in esecuzione si può usare il comando `renice`. In questo caso la priorità (o meglio il valore di *nice*) deve essere indicato con il suo valore numerico come primo argomento; si possono specificare il processo (o i processi) a cui applicare il cambiamento con gli argomenti successivi.

Il comando `renice` interpreta tutti gli argomenti successivi al primo come indicazione dei processi di cui si vuole cambiare la priorità. Il default è considerare gli argomenti come valori del PID di un processo, cosa che può essere indicata esplicitamente usando l'opzione `-p`; con l'opzione `-u` si possono selezionare tutti i processi di un singolo utente, specificato per nome o tramite il suo identificativo. Infine con `-g` si possono indicare tutti i processi di uno stesso gruppo (che in genere, come vedremo in sez. 2.1.2, corrispondono ai comandi eseguiti su una stessa riga di shell) specificandone il numero di *process group*.

Al solito si applica la restrizione che solo l'amministratore può applicare valori negativi, ed aumentare così la priorità di un processo. Si tenga conto inoltre che a differenza di `nice`, i valori di priorità impostati con `renice` sono assoluti, ed indicano il valore di *nice* da impostare. Una volta che si è diminuita la priorità di un processo aumentandone il valore di *nice* un utente normale in teoria non potrebbe tornare indietro, ma con le versioni recenti del commando questo non è più vero.[55]

1.4 Il controllo degli accessi

Come già detto e ripetuto più volte, Linux è nato come sistema multiutente e nella sua architettura è nativa la possibilità di avere utenti diversi che lavorano sulla stessa macchina. Questo ovviamente comporta la necessità di un meccanismo di controllo degli accessi, che permetta di restringere le capacità dei singoli utenti in maniera che questi non possano recare danni (volontariamente o involontariamente che sia) agli altri o al sistema. Tratteremo qui le basi dei meccanismi di controllo degli accessi, mentre la gestione degli utenti sarà affrontata più avanti, in sez. 4.3.

1.4.1 L'identificazione di utenti e gruppi

Essendo nato come sistema multiutente, ogni kernel di tipo unix-like come Linux deve fornire dei meccanismi di identificazione dei vari utenti, sulla base dei quali poi possa venire imposto un adeguato controllo degli accessi e delle operazioni che questi possono eseguire all'interno del sistema.

[55] questo avveniva fino alla versione 2.2.4 della *glibc*, quando è stata modificata la funzione di libreria `nice`, che non chiama più la corrispondente *system call* del kernel che ha questo limite, ma usa altre funzioni che consentono un valore assoluto, così che anche il comando `renice` diventa capace di diminuire anche per un utente normale un valore di *nice* precedentemente aumentato, ma mai al di sotto di quello di partenza.

La struttura di sicurezza tradizionale di un sistema unix-like è estremamente semplice, tanto da essere in certi casi considerata troppo primitiva.[56] Essa prevede una distinzione fondamentale fra l'amministratore del sistema, per il quale non esiste nessuna restrizione, e tutti gli altri utenti, per i quali invece vengono effettuati vari controlli, a seconda delle operazioni che vengono richieste. Tradizionalmente l'amministratore viene associato all'username "*root*", tanto che questo nome spesso viene usato come sinonimo di amministratore.

Come già accennato in sez. 1.3.2 all'interno del sistema il kernel identifica ogni utente con un numero identificativo, chiamato *user ID* o UID. Questo è nullo per l'amministratore e diverso da zero per tutti gli altri utenti. I controlli vengono effettuati dal kernel sulla base di questo numero, e *se* è diverso da zero viene verificato se corrisponde a quello per cui la richiesta è consentita, e nel caso si nega o concede l'accesso. Si tenga presente che quello che conta è comunque l'*user ID*, non il nome utente, questo infatti è solo un'etichetta che potrebbe essere qualunque (torneremo sull'argomento in sez. 4.3).

Si noti inoltre il precedente *se*, qualora l'*user ID* del processo sia nullo il controllo non viene neanche effettuato: è per questo che *root* è sempre in grado di compiere qualunque operazione, ed è per questo che occorre evitare assolutamente l'uso di *root* per qualunque compito che non sia strettamente connesso all'amministrazione del sistema: un comando sbagliato che sarebbe innocuo se dato da un utente normale potrebbe distruggere completamente il sistema se dato da *root*.

Tutti i sistemi unix-like prevedono una procedura di autenticazione che permette di riconoscere l'utente che si collega al sistema. Questa nella sua forma più elementare, per l'accesso da una *console*[57] testuale, viene effettuata dal programma `login`, che in base al nome che identifica un utente di fronte al sistema (il cosiddetto *username*, o *nome di login*) richiede l'immissione di una password ad esso associata, che permette di verificarne l'identità.

Come vedremo con maggiori dettagli in sez. 4.3, dove tratteremo la configurazione della procedura di login e le modalità per gestire utenti e gruppi, ciascun *username* è abbinato ad un *user ID*,[58] che sarà quello che verrà utilizzato dal kernel per tutti i programmi che l'utente lancerà una volta entrato nel sistema. I comandi invece, per una maggiore leggibilità, riportano l'identità dell'utente utilizzando il suo *username*, molti di essi comunque (come `ls`) supportano l'uso di opportune opzioni (nel caso `-n`) per stampare il valore numerico dell'*user ID* al posto dell'*username*.

Gli utenti poi possono venire raggruppati in *gruppi*; come per gli utenti ogni gruppo ha un nome (detto *groupname*) ed un corrispondente identificatore numerico, il *group ID* o GID. Inoltre ad ogni utente è sempre associato almeno un *gruppo*, detto *gruppo primario* o *gruppo principale*, che è quello che viene usato anche come *group ID* associato al processo di cui abbiamo parlato in sez. 1.3.2.

[56] sono previste estensioni specifiche, come quelle di SELinux (incorporate nel kernel a partire dalle versioni di sviluppo 2.5.x), che implementano il *Mandatory Access Control* e la capacità di definire in maniera molto più dettagliata i privilegi di accesso, sottoponendo anche l'amministratore ad ulteriori restrizioni; questo necessita ovviamente di un bel po' di lavoro amministrativo in più per cui non sono utilizzate molto spesso.

[57] un tempo si indicava con questo nome il terminale privilegiato da cui si potevano eseguire i compiti amministrativi, in genere collegato direttamente al computer stesso; oggi in pratica i compiti amministrativi si possono effettuare da qualunque terminale, per cui *console* è diventato sostanzialmente un sinonimo di terminale.

[58] questa funzionalità viene fornita direttamente dalle librerie fondamentali del sistema (le *glibc*) tramite il cosiddetto *Name Service Switch* (vedi sez. 4.3.6).

Di norma si fa sì che questo gruppo contenga come membro solo l'utente in questione e abbia nome uguale all'username, per dare un gruppo specifico ai file personali dell'utente dato che questo è il gruppo che viene usato nella creazione di nuovi file (torneremo sull'argomento in sez. 1.4.2). In generale però un utente può appartenere ad un numero qualunque di ulteriori gruppi, detti *gruppi ausiliari*; diventa così possibile permettere l'accesso a delle risorse comuni a tutti gli utenti che fanno parte dello stesso gruppo.

Ogni processo, quando viene lanciato, eredita dal padre l'informazione relativa all'utente per conto del quale viene eseguito e di tutti i gruppi a cui questo appartiene. In questo modo una volta entrati nel sistema tutti i programmi verranno eseguiti per conto dell'utente che ha effettuato la procedura di login. Il sistema può così provvedere al controllo degli accessi, e porre una serie di limitazioni a quello che l'utente può fare, impedendo l'esecuzione di tutte le operazioni non consentite. Si evita così che utenti diversi possano danneggiarsi fra loro o danneggiare il sistema.

In realtà, come già accennato in sez. 1.3.2, il kernel mantiene diverse versioni di questi identificatori; la trattazione esatta di tutti i particolari va al di là dello scopo di questo testo,[59] per quanto ci interessa qui basta citare le due versioni principali, quella degli indicatori cosiddetti *effettivi* (cioè l'*effective user ID* e l'*effective group ID*) che vengono usati per le verifiche sul controllo d'accesso e quelli cosiddetti *reali* (cioè il *real user ID* ed il *real group ID*) che identificano chi ha lanciato il processo. In generale questi due gruppi di identificatori sono sempre identici fra loro, a meno che non si sia usato un programma con i permessi speciali *set user ID* o *set group ID*, come vedremo più avanti in sez. 1.4.2.

1.4.2 I permessi dei file

La gestione dei permessi dei file in un sistema unix-like è tutto sommato abbastanza elementare; esistono estensioni[60] che permettono di renderla più sottile e flessibile, ma nella maggior parte dei casi il controllo di accesso standard è più che sufficiente. In sostanza per il sistema esistono tre livelli di privilegi:

- i privilegi dell'utente, indicati con la lettera `u`, (dall'inglese *user*).
- i privilegi del gruppo, indicati con la lettera `g`, (dall'inglese *group*).
- i privilegi di tutti gli altri, indicati con la lettera `o`, (dall'inglese *other*).

e tre permessi base:

- permesso di lettura, indicato con la lettera `r`, (dall'inglese *read*).
- permesso di scrittura, indicato con la lettera `w`, (dall'inglese *write*).
- permesso di esecuzione, indicato con la lettera `x`, (dall'inglese *execute*).

il cui significato è abbastanza ovvio.[61]

Ogni file è associato ad un utente, che è detto *proprietario* del file e ad un gruppo (detto a sua volta *gruppo proprietario*), per ciascuno dei tre livelli di privilegio (utente, gruppo e

[59] come già detto gli interessati possono trovare una trattazione dell'argomento nella sezione 3.3 di [GaPiL].

[60] come le ACL (*Access Control List*), introdotte ufficialmente a partire dai kernel 2.6, o altri meccanismi specialistici ancora più sofisticati (come SELinux, cui abbiamo già accennato).

[61] almeno fintanto che si fa riferimento ad un file normale, vedremo a breve che le cose cambiano quando si tratta di directory, link simbolici o dispositivi.

altri) è possibile specificare uno dei tre permessi (lettura, scrittura, esecuzione). Questi vengono riportati nella versione estesa dell'output del comando `ls`:

```
piccardi@anarres:~/GaPiL$ ls -l
total 2996
-rw-r--r-- 1 piccardi piccardi     67 Feb  9 17:01 AUTHORS
-rw-r--r-- 1 piccardi piccardi   8436 Feb  9 17:01 biblio.bib
-rw-r--r-- 1 piccardi piccardi  29247 Feb  9 17:01 build.tex
-rw-r--r-- 1 piccardi piccardi   7286 Feb  9 17:01 ChangeLog
-rw-r--r-- 1 piccardi piccardi  29941 Feb  9 17:01 errors.tex
-rw-r--r-- 1 piccardi piccardi  17277 Feb  9 17:01 fdl.tex
-rw-r--r-- 1 piccardi piccardi 297727 Feb  9 17:01 fileadv.tex
-rw-r--r-- 1 piccardi piccardi 415730 Feb  9 17:01 filedir.tex
-rw-r--r-- 1 piccardi piccardi 205773 Feb  9 17:01 fileio.tex
-rw-r--r-- 1 piccardi piccardi   5117 Feb  9 17:01 gapil.tex
drwxr-xr-x 3 piccardi piccardi   4096 Feb  9 17:01 htgapil
-rw-r--r-- 1 piccardi piccardi   7592 Feb  9 17:01 htgapil.cfg
-rwxr-xr-x 1 piccardi piccardi    175 Feb  9 17:01 htmakeindex.sh
drwxr-xr-x 3 piccardi piccardi   4096 Feb  9 17:01 html
-rwxr-xr-x 1 piccardi piccardi   2256 Feb  9 17:01 htmlize.sh
drwxr-xr-x 3 piccardi piccardi   4096 Feb  9 17:01 img
-rw-r--r-- 1 piccardi piccardi  84338 Feb  9 17:01 intro.tex
...
```

che ci mostra nella prima colonna i permessi secondo lo schema riportato in fig. 1.5. La lettura dell'output di questo comando ci permette di vedere che i sorgenti dei programmi contenuti nella directory in oggetto hanno quelli che sono in genere i permessi di default per i file di dati, si ha cioè il permesso di scrittura solo per il proprietario, quello di lettura per tutti quanti (proprietario, gruppo e altri) e l'assenza completa del permesso di esecuzione.

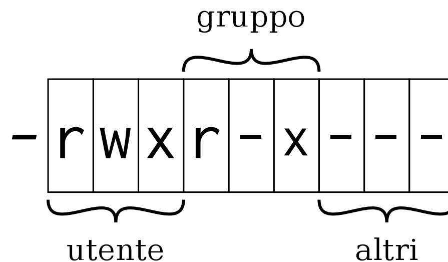


Figura 1.5: Legenda dei permessi dell'output di `ls`.

Nel caso di un programma o uno script (tratteremo gli script in sez. 2.1.5) invece i permessi di default sono diversi, come per `htmlize.sh` nell'esempio precedente, ed oltre a quelli presenti per i normali file di dati si ha anche il permesso di esecuzione, che è abilitato per il proprietario, per il gruppo e per tutti gli altri.

In generale un utente potrà effettuare solo le azioni per le quali ha i permessi, nel caso dei permessi appena illustrati questo comporta che un utente può leggere i file di un altro ma non può modificarli o cancellarli. In realtà il meccanismo del controllo è abbastanza peculiare, e questo,

se non capito fino in fondo, può dar luogo a qualche confusione.[62] La verifica dei permessi di accesso ad un file prevede infatti i seguenti passi, eseguiti esattamente in questa sequenza:

1. se il processo viene eseguito dall'amministratore l'accesso viene sempre garantito, senza che sia eseguito nessun controllo;
2. se il processo viene eseguito dal proprietario del file allora:
   - vengono controllati i permessi corrispondenti all'utente (i primi tre di fig. 1.5), e se questi sono assegnati per l'operazione richiesta, l'accesso è consentito;
   - altrimenti l'accesso è negato;
3. se il processo viene eseguito da un utente che appartiene al gruppo proprietario del file allora:
   - vengono controllati i permessi corrispondenti al gruppo (i centrali in fig. 1.5), e se questi sono assegnati per l'operazione richiesta, l'accesso è consentito;
   - altrimenti l'accesso è negato;
4. se il processo viene eseguito da un qualunque altro utente allora:
   - vengono controllati i permessi corrispondenti a tutti gli altri (gli ultimi di fig. 1.5), e se questi sono assegnati per l'operazione richiesta, l'accesso è consentito;
   - altrimenti l'accesso è negato.

Si noti bene come in questa sequenza di controlli, una volta che la decisione riguardo l'accesso è presa, i passi successivi non vengono più eseguiti. Tutto ciò significa ad esempio che se un utente non ha il permesso di scrittura per un proprio file, questi non potrà scriverci, anche qualora il permesso di scrittura per il gruppo o per tutti gli altri fossero garantiti.

Questo avviene perché il permesso relativo all'utente proprietario viene controllato per primo, e se l'accesso viene negato i permessi successivi non vengono controllati. Questo spiega anche perché il permesso indicato con "`o`" si chiama *per tutti gli altri*, e non semplicemente per tutti: esso infatti verrà applicato soltanto per gli utenti che non sono il proprietario e non appartengono al gruppo proprietario.

Vale la pena anche sottolineare di nuovo come i controlli siano eseguiti per tutti gli utenti eccetto l'amministratore, che non è soggetto a nessun tipo di restrizione e può quindi eseguire qualunque operazione. In genere ogni distribuzione fa sì che tutti i file di configurazione ed i programmi installati nel sistema appartengano a *root*, cosicché diventa impossibile per un utente normale poter danneggiare (accidentalmente o meno) gli stessi.[63]

Fin qui si è dato abbastanza per scontato il significato dei tre permessi di lettura, scrittura ed esecuzione, fra l'altro facendo riferimento solo a file normali (cioè ai *regular file* di tab. 1.1). In realtà anche in questo caso il funzionamento del sistema ha alcune sottigliezze che occorre approfondire. Anzitutto bisogna chiarire che i suddetti permessi si applicano al *contenuto* del file: il permesso di lettura significa che se ne può leggere il contenuto, quello di scrittura che se ne può modificare il contenuto, e quello di esecuzione che se ne può eseguire il contenuto (il che si traduce nel poter lanciare il programma).

[62] per una trattazione dettagliata si consultino sez. 3.2 e 4.4 di [GaPiL].

[63] questa è una delle ragioni per cui i virus sono molto più difficili da fare con Linux: non essendo possibile ad un utente normale modificare i file di sistema, un virus potrà al più "*infettare*" i file di quell'utente, cosa che rende molto più difficile la sua diffusione.

Il fatto che i permessi facciano riferimento al contenuto di un file ha delle precise conseguenze, ad esempio avere il permesso di scrittura su un file non significa che se ne possano cambiare i permessi, perché questi, come accennato in sez. 1.2.2, sono mantenuti nell'*inode* e non hanno perciò nulla a che fare con il suo contenuto. Il sistema infatti prevede che solo il proprietario di un file possa modificarne i permessi (con il comando `chmod`, che vedremo in sez. 1.4.3) e la cosa è del tutto indipendente dai permessi che detto file può avere, il che consente al proprietario, anche qualora non avesse un certo permesso su un file (come nell'esempio precedente per la scrittura) di riassegnarselo.

I permessi inoltre hanno un significato leggermente diverso quando si applicano agli altri tipi di file illustrati in tab. 1.1. Nel caso di file di dispositivo, *fifo* e *socket* ad esempio il permesso di esecuzione è ignorato, mentre per i link simbolici tutti i permessi sono totalmente ignorati (è questo il motivo per cui un collegamento simbolico li presenta come tutti attivi) e quelli che vengono presi in considerazione sono quelli del file a cui il collegamento fa riferimento.

Le differenze maggiori però si ritrovano per le directory. Alcune di queste differenze in realtà sono minori di quel che possono sembrare ed il significato dei permessi si può capire subito se si tiene presente che, come illustrato in sez. 1.2.2, le directory non sono altro che dei file di tipo speciale il cui contenuto è semplicemente una lista di nomi associati ad un riferimento ai rispettivi *inode*. Questo ci dice che il permesso di lettura per una directory si traduce semplicemente nella possibilità di leggere, e quindi visualizzare, la lista dei file in essa contenuti.

Un po' meno intuitivo, se non altro nei suoi effetti, è il permesso di scrittura per una directory. Di nuovo questo significa semplicemente che si può scrivere nel suo contenuto, il che si traduce nel fatto che si può aggiungere una voce alla lista, per creare un file, o toglierla, per cancellarlo.[64] È in particolare questa seconda possibilità che normalmente lascia perplesso chi si accosta le prime volte a Linux perché, a differenza di altri sistemi operativi, non esiste un permesso specifico per cancellare un file. Per tale operazione infatti è sufficiente avere il permesso di scrittura sulla directory che lo contiene, infatti è questa che viene modificata, e si noti come l'operazione sia del tutto indipendente da quelli che possono essere i permessi del file, che nel caso non contano niente.

Il più complesso da capire resta comunque il permesso di esecuzione che normalmente, come per la directory `img` dell'esempio precedente, è sempre presente. Le directory non sono programmi ed ovviamente "*eseguire*" una lista di file non ha nessun significato. Il kernel perciò usa questo permesso dandogli un significato particolare, che è quello di poter "*attraversare*" la directory nella risoluzione del nome di un file.

Se cioè manca il permesso di esecuzione, indipendentemente da quali possano essere i permessi impostati sui file e le directory in essa contenuti, questi non saranno più accessibili, dato che non sarà possibile attraversarla per raggiungerli, il che significa che eliminando il permesso di esecuzione su una directory si preclude l'accesso all'intero ramo dell'albero dei file posto al di sotto di essa. Si tenga presente che tutto ciò resta valido anche in presenza del permesso di lettura sulla directory, che significa solo che è possibile elencarne il contenuto.

Ci si può chiedere a cosa serva questa separazione fra il permesso di vedere il contenuto di una directory ed il permesso di poterla attraversare; esistono però casi in cui si può voler consentire a degli utenti (in genere appartenenti ad un certo gruppo) di accedere a dei file (di cui conoscono il nome) all'interno di una directory condivisa, ma non si vuole che questi possano vedere se ci

[64] si ricordi infatti che, come illustrato in sez. 1.2.2, la cancellazione di un file consiste semplicemente nella rimozione del riferimento al relativo *inode* presente nella directory.

sono anche file di altri. Un esempio di tutto ciò è la directory `/etc/ssl/private`, i cui permessi sono:

```
piccardi@anarres:~$ ls -ld /etc/ssl/private/
drwx--x--- 2 root ssl-cert 1024 2007-10-18 12:54 /etc/ssl/private
```

che permette agli utenti del gruppo `ssl-cert` di accedere ai file creati sotto tale directory ma non di vederne il contenuto. La directory è creata dal pacchetto `openssl` per contenere le chiavi dei certificati digitali,[65] e viene impostata con questi permessi per far sì che un utente possa accedere alle proprie chiavi, ma non possa sapere se ve ne sono anche di altri utenti.

Fino ad adesso si sono trattati soltanto i permessi ordinari, a questi però in tutti i sistemi unix-like sono stati aggiunti tre *permessi speciali* che permettono di rendere più flessibile una infrastruttura che, se fosse stata limitata soltanto a quegli ordinari, sarebbe stata troppo limitata. Dal punto di vista del kernel ciascun permesso corrisponde ad un bit in una apposita variabile mantenuta nell'*inode* (vedi sez. 1.2.2) del file. I bit usati per i permessi sono in tutto 12; i primi nove sono quelli già illustrati in fig. 1.5, gli altri tre vengono chiamati col nome assegnato al rispettivo bit, e cioè: *suid bit*, *sgid bit* e *sticky bit*.

Questi tre permessi aggiuntivi nascono per modificare il comportamento del sistema nell'esecuzione dei programmi. Per questo normalmente sono sempre abbinati al permesso di esecuzione,[66] ed hanno effetto solo se vengono applicati a dei binari eseguibili (nel caso siano presenti su uno script come misura di sicurezza vengono ignorati). I più importanti sono i primi due che servono per consentire ad un programma di eseguire delle operazioni che nel caso generico sarebbero riservate al solo amministratore.

Il sistema infatti quando esegue un programma lo fa sempre con i permessi dell'utente che lo ha lanciato, il che significa, per riprendere quanto illustrato in sez. 1.4.1, che gli identificativi *real* ed *effective* di utente e gruppo del nuovo processo coincidono con quelli del processo che lo ha creato. Per consentire ad un processo lanciato da un utente normale di eseguire degli accessi che i permessi normalmente gli impedirebbero, è possibile, impostando rispettivamente il *suid bit* o lo *sgid bit* su un programma, far sì che quest'ultimo venga eseguito rispettivamente con i privilegi dell'utente e del gruppo a cui appartiene. Questo è ovviamente pericoloso, dato che questi privilegi possono essere maggiori di quelli di un utente normale, è pertanto assolutamente inopportuno impostare questi permessi su un programma senza sapere esattamente cosa si sta facendo.

Tutto questo viene realizzato dal kernel che quando trova questi permessi attivi per un programma lo pone in esecuzione impostando per il corrispondente processo l'*effective user ID* (se *suid*) o l'*effective group ID* (se *sgid*) rispettivamente a quello corrispondente all'utente o al gruppo proprietario del file. Dato che gli identificativi del gruppo *effective* sono quelli usati nel controllo degli accessi, tutto ciò fa sì che il programma risulti avere i privilegi di tale utente o gruppo. Gli identificativi del gruppo *real* però resteranno gli stessi, e cioè quelli dell'utente che ha eseguito il programma, il quale sarà in grado di accorgersi di essere stato lanciato da un utente normale.

Con l'uso di questi permessi si possono far eseguire anche ad un utente normale delle operazioni che nel caso generico sarebbero riservate al solo amministratore. Questo ad esempio è il

[65] i dettagli di SSL vanno al di là di quanto sia possibile affrontare qui, per una trattazione dell'argomento si veda sez. 2.1 di [SGL].

[66] con l'eccezione del cosiddetto *mandatory locking*, una estensione ripresa da SysV su cui potete trovare maggiori dettagli in sez. 10.1.5 di [GaPiL], che viene attivato impostando lo *sgid bit* su un file non eseguibile.

modo in cui funziona il comando `passwd`, che ha il bit *suid bit* impostato ed appartiene all'amministratore. Se usato direttamente da questi il programma (che verifica quale sia l'utente che l'ha posto in esecuzione controllando il *real user ID*) non pone restrizioni e permette di cambiare la password a qualunque utente. Se invece viene usato da un utente normale il programma consente ancora di modificare la password, ma solo la propria e fornendo prima quella corrente. Per poter eseguire la modifica della password `passwd` necessita dell'accesso in scrittura al file dove questa viene memorizzata (che, vedi sez. 4.3.2, è di proprietà dell'amministratore) e questo viene garantito appunto grazie all'uso del *suid bit*.

Per i file lo *sticky bit* al giorno d'oggi è sostanzialmente inutilizzato e su Linux viene ignorato. Veniva usato nelle prime versioni di Unix, quando memoria e disco erano risorse molto scarse e gli algoritmi per la memoria virtuale poco raffinati, per dare ad un programma un accesso privilegiato all'uso della *swap*. Per i programmi con questo permesso attivo (era usato per quelli di uso più frequente) il codice veniva mantenuto in permanenza nella *swap* per diminuire il tempo di caricamento, cosa da cui deriva il nome *sticky*. Oggi tutto ciò non è più necessario, per cui detto permesso viene ignorato, ha assunto però, come vedremo a breve, un significato speciale per le directory.

Al contrario di quanto avviene con i permessi normali, i permessi speciali non hanno a disposizione una propria posizione nell'output di `ls`, ma quando attivati cambiano il valore della lettera associata al permesso di esecuzione. In particolare se il *suid bit* o lo *sgid bit* sono attivi verrà rispettivamente mostrata una "`s`" nella posizione del permesso di esecuzione per utente e gruppo, o una "`S`" qualora il corrispondente permesso di esecuzione non sia attivo. Lo *sticky bit* invece modifica la lettera che indica il permesso di esecuzione per tutti in una "`t`" che come per i precedenti diventa una "`T`" qualora il corrispondente permesso di esecuzione non sia attivo. Nel caso di `passwd` avremo allora:

```
piccardi@anarres:~$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 29036 2007-10-30 19:32 /usr/bin/passwd
```

Abbiamo visto come i permessi speciali consentano di modificare il comportamento del kernel nell'esecuzione dei programmi, essi però non hanno nessun effetto su file di dispositivo, *fifo* o *socket*, per i quali vengono ignorati. Per le directory invece il discorso è diverso, il *suid bit* viene sempre ignorato, ma lo *sgid bit* e lo *sticky bit* vengono utilizzati, ed assumono un significato particolare.

Lo *sgid bit* viene utilizzato per modificare il comportamento ordinario del kernel nella creazione di nuovi file. Il sistema infatti prevede che quando un processo crea un nuovo file o directory, file di dispositivo, ecc. a questo vengano assegnati come proprietario e gruppo proprietario rispettivamente *user ID* ed *group ID* del processo stesso, in cui quest'ultimo corrisponde di solito al gruppo principale dell'utente. Se però si è attivato lo *sgid bit* nella directory in cui si va a creare il file a questi sarà assegnato come gruppo proprietario quello della directory stessa e non quello del processo. In questo modo si può far sì che il gruppo proprietario di una directory si propaghi automaticamente a tutti i file che vengono creati al suo interno.

Anche lo *sticky bit* assume un significato particolare per le directory: se esso è presente allora un file potrà essere cancellato solo dall'utente a cui appartiene o dal proprietario della directory stessa, anche qualora si abbia su di essa il permesso di scrittura. Questa ad esempio è la modalità in cui viene protetta la directory `/tmp`, i cui permessi sono i seguenti:

```
piccardi@anarres:~$ ls -ld /tmp
drwxrwxrwt 18 root root 155648 2008-01-08 18:38 /tmp
```

si noti come questi permessi consentono a chiunque scrivere sulla directory `/tmp` per crearvi i propri file, come è dovuto, essendo questa la directory messa a disposizione degli utenti per la creazione di file temporanei come abbiamo visto sez. 1.2.3.

Il permesso di scrittura consentirebbe anche la cancellazione arbitraria degli stessi file, ma la presenza contemporanea dello *sticky bit* fa sì che un utente sia in grado di cancellare solo i file che ha creato lui senza poter interferire con quelli creati dagli altri utenti. Questa è una misura necessaria, perché si deve sempre ricordare che, a parte questa eccezione, qualunque siano i permessi dati ad un file sarà sempre possibile cancellarlo se si ha il permesso di scrittura sulla directory che lo contiene.

1.4.3 La gestione dei permessi dei file

Come accennato in precedenza ciascun permesso corrisponde ad uno specifico bit di una variabile che viene mantenuta nell'*inode* del file insieme a tutte le altre proprietà dello stesso. Assegnare un permesso significa attivare il bit ad esso associato in questa variabile. La disposizione completa dei bit dei permessi, compresi quelli relativi ai permessi speciali, è riportata in fig. 1.6.

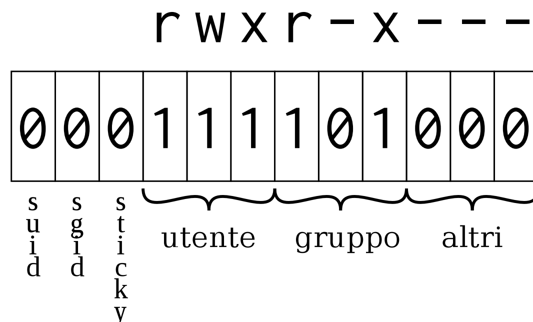


Figura 1.6: Schema dei bit dei permessi mantenuti nell'*inode*.

Sfruttando questa disposizione è possibile utilizzare una notazione compatta che esprima lo stato dei permessi usando il valore di questa variabile. Dato che i permessi sono raggruppati a blocchi di tre bit, risulta naturale esprimere questo valore in notazione ottale,vale a dire con numeri in base 8, con le cifre che vanno da 0 a 7, in modo che ciascuna cifra corrisponda ad un gruppo di permessi. Se si fa riferimento allo schema di fig. 1.6, in questo modo si avrà che, partendo dalla meno significativa, la prima cifra esprime il valore dei permessi per gli altri, la seconda quello dei permessi per il gruppo, e la terza quello dei permessi per il proprietario. Con questa notazione allora i permessi presenti nel caso illustrato in fig. 1.6 si possono indicare con la cifra `750`.

Qualora serva specificare anche uno dei permessi speciali, occorrerà usare anche la quarta cifra, se di nuovo si fa riferimento allo specchietto di fig. 1.6, e lo si confronta con i permessi del comando `passwd` che si sono visti in precedenza, considerato che questo ha attivo il bit *suid*, si otterrà che questi si possono esprimere con il valore `4755`. Per la sua compattezza questa notazione, nonostante costringa l'utente a fare un po' di conti in binario ed ottale, è molto utilizzata, ed una volta che ci si è abituati, finisce in genere anche col risultare più immediata

di quella ordinaria, che, come vedremo più avanti, è basata sull'uso delle lettere che indicano livello e tipo riportate all'inizio di sez. 1.4.2.

Prima di illustrare il funzionamento dei comandi che consentono la manipolazione dei permessi dei file conviene esaminare meglio le modalità con cui questi vengono assegnati ai nuovi file e directory che vengono creati. Abbiamo visto in sez. 1.4.2 come vengono assegnati di default l'utente ed il gruppo proprietario, ci resta da vedere come vengono assegnati i permessi. Questi dipendono in maniera diretta dalle modalità con cui i programmi creano i file, e da quali funzioni di sistema essi utilizzano.

Il kernel prevede comunque una modalità per controllare i permessi assegnati di default. Il sistema prevede che si parta da uno stato in cui alla creazione di un nuovo file i permessi di lettura e scrittura sono attivati per *tutti*, dove con tutti qui si vuole intendere sia il proprietario, che il gruppo, che tutti gli altri. Si parte cioè da dei permessi espressi in forma numerica dal valore ottale di `666`. Per la creazione di una nuova directory invece lo stato di partenza prevede anche l'assegnazione del permesso di esecuzione, con un valore corrispondente a `777`.

Ovviamente lasciare aperti a chiunque i permessi di scrittura non è cosa molto saggia dal punto di vista della sicurezza. Per questo motivo il kernel mantiene per ogni processo una proprietà, chiamata *umask*, che viene ereditata nella creazione di un processo figlio. La *umask* specifica una maschera di bit che serve ad indicare quali permessi devono essere rimossi alla creazione di un nuovo file o directory. Il formato in cui si specifica la maschera è lo stesso utilizzato per i bit dei permessi ed illustrato in fig. 1.6, solo che in questo caso il valore indicherà quali sono i bit da togliere.

In genere la *umask* viene impostata negli script eseguiti al *login* grazie al comando `umask`. Dato che il valore viene ereditato nella creazione dei processi figli è sufficiente farlo all'avvio perché quanto scelto sia mantenuto anche in seguito. Un utente può comunque modificare il valore di questo parametro invocando direttamente il comando `umask`[67] per impostare un nuovo valore, da passare come argomento.

Il valore dell'argomento di `umask` può essere specificato sia in forma numerica (con un valore ottale) che in forma simbolica, con una notazione identica a quella di `chmod`, che tratteremo a breve. In questo secondo caso però si dovranno indicare quali sono i permessi da *conservare*, se si usa un valore numerico infatti questo viene impostato direttamente come *umask* del processo, per cui vi si indicano i permessi da togliere, il valore simbolico invece viene usato per indicare quali permessi devono restare. Se invocato senza argomenti il comando permette di visualizzare il valore corrente della *umask*, e con l'opzione `-S` ne stampa il valore in forma simbolica anziché numerica. Di norma un utente personalizza questo valore negli script di avvio della shell (vedi sez. 2.1.5).

Il valore di default per la *umask* usato dalla gran parte delle distribuzioni è di `022`; questo significa che i permessi di scrittura relativi al gruppo e a tutti gli altri sono cancellati alla creazione di un nuovo file o directory: i nuovi file saranno creati cioè con permessi `644` (lettura e scrittura per il proprietario e sola lettura per il gruppo e gli altri), mentre le nuove directory saranno create con permessi `755` (tutti i permessi per il proprietario ma solo lettura e attraversamento per il gruppo e gli altri). Se ad esempio si fossero voluti cancellare completamente i permessi per tutti gli altri, si sarebbe dovuta usare una *umask* di `027`.

[67] si tenga presente che questo è un comando interno della shell (vedremo il significato di ciò in sez. 2.1.3), dato che l'invocazione di un altro programma non servirebbe a nulla, in quanto la *system call* che esegue l'operazione opera solo sul processo corrente.

Il comando che permette di modificare direttamente i permessi di file e directory è `chmod`. Nella sua forma più elementare esso prende come argomenti il valore numerico dei permessi (espresso in forma ottale) seguito dal nome o dalla lista dei nomi di file e directory cui applicarli. Come accennato però le operazioni di modifica dei permessi possono essere espresse anche in forma simbolica, utilizzando un formato comune che viene usato da tutti i comandi che operano sui permessi. Questo formato prevede che una operazione (o un valore di un permesso) possano essere specificati tramite una opportuna stringa di caratteri divisa in tre parti, nella forma:

```
[set][operator][permission]
```

La prima parte della stringa, *set*, specifica a quale gruppo di permessi si fa riferimento. Si può usare il carattere "`u`" (*user*) per indicare quelli relativi al proprietario, il carattere "`g`" (*group*) per indicare quelli relativi al gruppo ed il carattere "`o`" (*others*) per indicare tutti gli altri. Inoltre con il carattere "`a`" (*all*) si possono indicare in un colpo solo proprietario, gruppo e tutti gli altri. Se non si indica nulla si ha lo stesso effetto di "`a`", a meno che fra quelli indicati non vi siano permessi non consentiti dalla *umask*, che non verranno impostati. La notazione prevede anche che si possano usare più caratteri per specificare più insiemi a cui assegnare gli stessi permessi: ad esempio con `ug` si indicherà di operare sui permessi del proprietario e del gruppo proprietario.

La seconda parte della stringa, *operator*, indica il tipo di modifica che si vuole effettuare, questa prevede l'uso di un singolo carattere che serve appunto da *operatore*. I caratteri consentiti in questa parte sono tre: con "`+`" si richiede l'aggiunta di un permesso a quelli già presenti, con "`-`" si richiede la rimozione di un permesso da quelli presenti, mentre con "`=`" si indicano esattamente quali sono i permessi da assegnare, vale a dire che saranno assegnati esattamente quelli indicati, mentre quelli non indicati saranno rimossi.

L'ultima parte, *permission*, serve ad indicare quali permessi devono essere applicati (aggiunti, rimossi o impostati). Per i permessi ordinari si possono usare i caratteri "`w`" per il permesso di scrittura, "`r`" per il permesso di lettura e "`x`" per il permesso di esecuzione. Quando si opera sugli insiemi di proprietario e gruppo proprietario il carattere "`s`" permette di impostare i corrispondenti bit *suid* e *sgid*, infine con il carattere "`t`" si può impostare lo *sticky bit*.

Così ad esempio `a+x` aggiunge il permesso di esecuzione per tutti, `o-r` toglie il permesso di lettura per gli altri, `g+w` aggiunge il permesso di scrittura al gruppo proprietario, `u=rw` imposta i permessi di lettura e scrittura per il proprietario (eliminando un eventuale permesso di esecuzione) e `u+s` attiva il *suid bit*. Infine si tenga presente che si possono anche combinare insieme più espressioni come le precedenti, scrivendole in una sequenza separata da virgole: così ad esempio con `g+w,o-r` si abilita la scrittura per il gruppo proprietario e si disabilita la lettura per tutti gli altri.

Si ricordi che un utente può cambiare i permessi solo dei file che gli appartengono e che il permesso di scrittura in questo caso non significa nulla in quanto si applica solo al contenuto del file e non alle sue proprietà. Inoltre se si usa `chmod` su un collegamento simbolico l'operazione verrà eseguita sul file referenziato dal link,[68] e valgono quindi le credenziali di quest'ultimo, questo significa che anche se il link vi appartiene non potete cambiare i permessi del file a cui esso fa riferimento fintanto che anche questo non vi appartiene.

Il comando inoltre supporta l'uso dell'opzione `-R` che, quando lo si applica ad una directory, esegue la modifica dei permessi ricorsivamente per tutto il tratto di albero contenuto nella

[68] si ricordi che i permessi di un link non hanno alcun significato e pertanto non avrebbe senso cambiarli.

stessa. In questo caso però se nella sezione di albero ci sono dei collegamenti simbolici, questi verranno ignorati ed i file a cui fanno riferimento resteranno immutati. La versione GNU/Linux del comando supporta anche l'opzione `--reference=FILE` che permette di fare riferimento ai permessi di un altro file. Per la descrizione completa del comando e di tutte le opzioni al solito si faccia riferimento alla pagina di manuale, accessibile con `man chmod`.

Oltre ai permessi può essere necessario anche cambiare proprietario e gruppo di un file (o di una directory). Per il cambiamento del proprietario il comando da utilizzare è `chown`, il comando prende come primo argomento il nome utente (o il valore numerico dell'*user ID*) a cui assegnare la proprietà del file e come argomenti successivi la lista dei file su cui operare. Il comando è comunque in grado di cambiare in contemporanea anche il gruppo proprietario specificando il primo argomento nella forma `utente:gruppo`, e viene supportata per compatibilità anche la forma obsoleta `utente.gruppo`.

Se invece si vuole cambiare solo il gruppo proprietario il comando da usare è `chgrp`, che prende di nuovo come primo argomento il nome del gruppo (o il valore numerico del *group ID*) seguito dalla lista dei file a cui applicare il cambiamento. I comandi `chown` e `chgrp` sono simili e prevedono opzioni analoghe: in particolare con `-R` si eseguono i cambiamenti ricorsivamente su tutto il contenuto di una directory e con `--reference` si può fare riferimento alle credenziali prese da un altro file.

Inoltre entrambi i programmi, quando usati su un link simbolico, operano sul file a cui questo fa riferimento, a meno che non si usi l'opzione `-h`, nel qual caso si opererà direttamente sul link. Questa operazione in realtà assume un significato soltanto se il link è una directory con lo *sticky bit* attivo, dove cambiare proprietario del link simbolico significa modificare l'identità dell'utente che può cancellarlo.

Infine si tenga conto che modificare il proprietario di file è una operazione privilegiata, che su Linux può essere eseguita solo dall'amministratore. Questa è una caratteristica di sicurezza mutuata da BSD che consente di evitare aggiramenti delle quote disco (vedi sez. 6.4); non è detto che sia presente su altri sistemi unix-like. In questo modo un utente normale non può assegnare ad un altro utente i suoi file, e può cambiare il gruppo proprietario soltanto per i file che gli appartengono, ed assegnare a questi ultimi soltanto gruppi a cui egli stesso appartiene. Solo l'amministratore ha la capacità piena di cambiare proprietario e gruppo di un file qualunque.

1.4.4 Altre operazioni privilegiate

Oltre al rispetto delle restrizioni imposte dai permessi dei file, il sistema sottopone gli utenti a controllo di accesso per una lunga serie di altre operazioni. Alcune le abbiamo già incontrate, come il cambiamento di proprietario o gruppo proprietario, secondo quanto appena visto in sez. 1.4.3, o la possibilità di inviare segnali ad un processo, che come accennato in sez. 1.3.3 per un utente normale è ristretta ai soli processi lanciati da lui, o la possibilità di aumentare la priorità di esecuzione di un processo che come visto in sez. 1.3.4 è possibile solo all'amministratore.

Oltre a queste ci sono una lunga serie di altre operazioni privilegiate che riguardano la rete (che vedremo meglio a partire da cap. 7), per cui solo l'amministratore può attivare e disattivare interfacce, modificare la tabella di instradamento dei pacchetti, applicare o rimuovere regole al sistema di *firewall*. All'amministratore inoltre è riservata l'allocazione delle prime 1024 porte usate dai protocolli UDP e TCP, per cui di norma soltanto lui è in grado di lanciare demoni che usano una di queste porte (dette per questo *riservate*) come un server web, di posta o un DNS.

Una particolare operazione riservata all'amministratore è quella della possibilità di usare la *system call* `chroot` ed il comando omonimo per eseguire processi con una directory radice diversa (si rammenti quanto accennato in sez. 1.3.2) in modo da restringerne l'accesso ad una sezione dell'albero dei file.

Uno degli impieghi più comuni è quello in cui è montato su una directory il disco di un altra installazione e si vuole lavorare come se si fosse all'interno di questa, cosa che si può fare usando appunto il comando `chroot`. Il comando prende come argomenti la directory da usare come radice, ed opzionalmente il programma da eseguire. Se non si specifica nessun programma verrà eseguita di default una shell, che ovviamente si troverà ad operare all'interno del tratto di albero specificato.

Il meccanismo del *chroot* viene spesso inteso come misura di sicurezza per la sua caratteristica di restringere il processo all'interno di un ramo dell'albero dei file in modo che esso non possa accedere alle altre parti che non lo riguardano. In realtà questo è vero soltanto se il processo non ha privilegi di amministratore (né modo di ottenerli) perché altrimenti diventa semplicissimo uscire dalla gabbia del *chroot*.[69]

Si tenga presente però che per poter funzionare un programma eseguito in un *chroot* dovrà trovare all'interno della relativa directory tutti i file (e programmi) che gli possono essere necessari,[70] per questo in genere i programmi che prevedono di operare in questa modalità richiedono anche l'approntamento di un ambiente appropriato all'interno della directory di *chroot*. In genere questa tecnica viene usata da alcuni demoni di rete per aumentare la propria sicurezza una volta che hanno rilasciato eventuali privilegi amministrativi. In definitiva un semplice *chroot* non è sufficiente per creare un ambiente virtuale ristretto ad una sottodirectory, per realizzarlo occorrono delle ulteriori restrizioni (presenti con estensioni come LXC o OpenVZ) la cui trattazione va al di là dello scopo di questo testo.

Una serie di ulteriori operazioni riservate all'amministratore sono: la possibilità di modificare il tempo sull'orologio di sistema (vedi sez. 2.4.3), la possibilità di riavviare o fermare il sistema (vedi sez. 5.3.5), la possibilità di caricare moduli del kernel (vedi sez. 5.2.5) e la possibilità di montare i filesystem. Quest'ultima operazione, anche se in parte può essere delegata agli utenti (come illustrato in sez. 5.1.3) resta comunque privilegiata in quanto l'esecuzione generica del comando `mount` con parametri qualunque può essere effettuata soltanto dall'amministratore.

Infine soltanto l'amministratore è in grado di creare i file speciali relativi ai dispositivi, cioè solo lui può usare il comando `mknod` per creare un file di dispositivo. Questo comando prende come argomenti il nome del file da creare seguito da una lettera, che indica il tipo di file speciale; questa può essere "`p`" per indicare una *fifo*, "`b`" per indicare un dispositivo a blocchi e "`c`" per indicare un dispositivo a caratteri. Qualora si crei un file di dispositivo (che sia a blocchi o a caratteri è lo stesso) devono poi essere specificati di seguito come ulteriori argomenti il *major number* ed il *minor number* [71] che lo identificano univocamente. Per una lista completa delle

[69] basta creare una sottodirectory, eseguire un `chroot` su di essa per avere la directory di lavoro corrente fuori dal *chroot*, per maggiori dettagli si veda sez. 4.5.5 di [GaPiL].

[70] questo è scontato nel caso del precedente esempio, molto meno se si vuole eseguire un programma qualunque in un ramo di albero qualunque.

[71] questi due numeri sono il meccanismo con cui storicamente vengono identificati i dispositivi all'interno del kernel (così come gli *inode* identificano un file); il nome sotto `/dev` è solo una etichetta, potrebbe essere qualunque (anche se poi molti script non funzionerebbero), quello che indica al kernel quale dispositivo usare quando si accede a quel file di dispositivo sono questi due numeri.

corrispondenze di questi numeri con i vari dispositivi si può fare riferimento al file `devices.txt` distribuito con i sorgenti del kernel nella directory `Documentation`.

Si noti che creare un file di dispositivo è una azione diversa dall'accedere al dispositivo sottostante, cosa che invece è regolata dai permessi di quest'ultimo come file. Pertanto se si è stati poco accorti e si è permesso agli utenti l'accesso in scrittura a `/dev/sda`, anche se questi non possono creare un file di dispositivo, potranno comunque scrivere nel suddetto disco e ad esempio saranno tranquillamente in grado di ripartizionarlo.

Si tenga presente inoltre che per un utente è sempre possibile creare una *fifo* usando il comando dedicato `mkfifo`; questo prende come argomento il nome della *fifo* e crea la relativa voce nel filesystem. Il comando supporta anche l'opzione `-m` che permette di specificare la maschera dei permessi (con la stessa sintassi di `chmod`) da applicare alla stessa. Non esiste invece un comando specifico per creare un *socket* su disco, i programmi che li usano li creano chiamando direttamente le *system call* necessarie.

Infine alcuni filesystem supportano delle ulteriori operazioni speciali sui file, la cui gestione non rientra nella normale politica dei controlli di accesso di sez. 1.4.2, ma viene effettuata attraverso quelli che vengono chiamati gli *attributi speciali*. Queste operazioni attengono ad una serie di capacità aggiuntive, fornite dai filesystem che le supportano, che possono essere attivate o meno per ciascun file.[72]

Alcuni di questi attributi, in particolare quelli che permettono di imporre delle speciali restrizioni all'accesso, possono essere impostati soltanto dall'amministratore, questo è il motivo per cui si è scelto di trattare questo argomento in questa sezione. Facendo riferimento a delle estensioni che non è detto siano presenti su un qualunque tipo di filesystem[73] gli attributi speciali dei file non possono essere visualizzati con il comando `ls`; per visualizzarli è stato creato un apposito comando, `lsattr`.

Il comando è analogo a `ls` sia per l'uso degli argomenti, che per l'uso delle opzioni `-R`, `-a` e `-d` che hanno lo stesso significato visto in tab. 1.3. Il comando stampa a video l'elenco dei file, preceduto dal relativo valore degli attributi speciali, pertanto se lo eseguiamo sui file di questo testo otterremo qualcosa del tipo:

```
piccardi@anarres:~/truedoc/corso$ lsattr *.tex
------------------- advadmin.tex
------------------- appendici.tex
------------------- config.tex
------------------- corso.tex
------------------- netadmin.tex
------------------- netbase.tex
------------------- netdns.tex
------------------- netinter.tex
------------------- netlpi.tex
------------------- ordadmin.tex
------------------- ringraziamenti.tex
------------------- shell.tex
------------------- stradmin.tex
------------------- struttura.tex
```

[72] come per tutte le altre proprietà di un file, anche la lista degli attributi speciali attivi viene mantenuta all'interno dell'*inode*.

[73] queste estensioni sono state introdotte con *ext2*, sono presenti nei successivi *ext3* ed *ext4*, ma sono supportate anche da altri filesystem.

che ci mostra come in questo caso nessun attributo speciale sia stato impostato. Quando uno di essi è attivo questo viene indicato nell'elenco dalla presenza della lettera che lo identifica (un elenco è riportato tab. 1.15) al posto del carattere "-". Alcune di queste proprietà dipendono comunque dal supporto nella versione specifica del filesystem che si sta utilizzando, e non è detto siano presenti in generale, inoltre alcune di queste, riportate nella seconda parte di tab. 1.15, possono soltanto essere rilette con `lsattr`.

Il comando che permette di impostare gli attributi speciali è `chattr`, questo prende come primo argomento una stringa che identifica quali attributi attivare o disattivare, e come argomenti successivi una lista di file. Il comando supporta anche l'opzione `-R` per eseguire ricorsivamente le operazioni quando nella lista dei file si è inclusa una directory, mentre se invocato con l'opzione `-V` diventa prolisso e descrive le operazioni effettuate.

La stringa che permette di specificare quali attributi attivare o disattivare è analoga a quella usata con `chmod` per i permessi, ciascun attributo è identificato da una lettera fra quelle riportate nella prima parte di tab. 1.15, in cui però non si sono menzionati alcuni attributi sperimentali, per i quali si rimanda alla lettura della pagina di manuale di `chattr`. Per attivare un attributo occorrerà farla precedere dal carattere "+", mentre se si vuole cancellare un attributo già impostato si dovrà usare il carattere "-". Come per `chmod` si possono specificare insieme anche più attributi, ed usare il carattere "=" per impostare la lista esatta di quelli che si vogliono impostare.

Attributo	**Significato**
`A`	blocca l'aggiornamento del tempo di ultimo accesso.
`a`	attiva il cosiddetto *append flag*, che consente la scrittura solo in coda al file; il contenuto corrente non può essere modificato e solo l'amministratore può attivare o disattivare questo attributo.
`c`	attiva la compressione trasparente del contenuto del file.
`D`	richiede che il contenuto della directory sia salvato su disco in maniera sincrona.
`d`	esclude il file dalla registrazione dei dati necessari ad eseguire il backup con `dump` (vedi sez. 4.1.3).
`i`	attiva il cosiddetto *immutable flag*, il file non può essere cancellato o rinominato, non si possono creare hard link, né modificarne il contenuto; solo l'amministratore può attivare o disattivare questo attributo.
`j`	richiede che tutti i contenuti del file siano prima scritti sul giornale (tratteremo il *journalling* dei filesystem in sez. 5.1.4); solo l'amministratore può attivare o disattivare questo attributo.
`s`	richiede che quando un file viene cancellato tutti i blocchi che contenevano i suoi dati siano riscritti su disco azzerandone il contenuto.[74]
`S`	richiede che il contenuto del file sia salvato su disco in maniera sincrona.
`u`	richiede che alla cancellazione il contenuto del file sia salvato in modo da poterne consentire il recupero.
`E`	segnala un errore nella compressione quando questa è stata attivata; l'attributo può solo essere letto da `lsattr`.
`I`	segnala che una directory viene indicizzata con gli *hash tree* (una funzionalità avanzata trattata in sez. 5.1.4); l'attributo può solo essere letto da `lsattr`.
`X`	permette di accedere direttamente al contenuto di un file compresso disabilitando la decompressione trasparente.
`Z`	indica che un file compresso è in uno stato inconsistente (*dirty*); l'attributo può solo essere letto da `lsattr`.

Tabella 1.15: Gli attributi speciali dei file.

Come accennato alcuni attributi, che solo l'amministratore può impostare, permettono di utilizzare delle speciali restrizioni di accesso, ad esempio con "`a`" si attiva l'*append flag* che consente soltanto di aggiungere dati ad un file mentre il contenuto corrente non può essere modificato;[75] questo può essere utilizzato per salvaguardare i file di log (vedi sez. 3.2.3) da eventuali modifiche, accidentali o meno, dato che questi sono in genere la prima cosa che un intruso cerca di modificare in caso di compromissione di un sistema. Con la presenza di questo attributo questo viene impedito fintanto che l'attributo non viene rimosso, purtroppo nelle operazioni ordinarie questo resta sempre possibile se si hanno i privilegi di amministratore.[76]

Un altro attributo che permette di attivare una protezione speciale è "`i`" che abilita il cosiddetto *immutable flag*, che finché attivo impedisce qualunque tipo di modifica al file.[77] Si noti che sia questo che l'*append flag* impongono le loro restrizioni anche alle operazioni richieste dall'amministratore stesso: questo significa ad esempio che un file immutabile potrà essere cancellato o modificato (anche dall'amministratore) soltanto se prima questi ne ha rimosso il relativo attributo. Di nuovo questa funzionalità permette di proteggere file essenziali in modo da impedirne ogni forma di modifica, anche se in maniera parziale visto che la protezione può essere rimossa dall'amministratore.

Potremo allora attivare alcuni di questi attributi (ad esempio l'*append flag* per un file di log, o l'*immutable flag* per `/etc/fstab`), in questo caso basterà eseguire i comandi:

```
root@anarres:~# chattr +a /var/log/auth.log
root@anarres:~# chattr +i /etc/fstab
```

e potremo controllare lo stato degli attributi ottenendo che:

```
root@anarres:~# lsattr /var/log/auth.log /etc/fstab
-----a----------- /var/log/auth.log
----i------------ /etc/fstab
```

a questo punto si potrà anche verificare che non è più possibile modificare `/etc/fstab`:

```
root@anarres:~# touch /etc/fstab
touch: cannot touch '/etc/fstab': Permission denied
```

nonostante il comando sia stato eseguito con privilegi di amministratore.

[74] di default infatti il kernel si limita a marcare i blocchi come liberi, ma non cancella il precedente contenuto, che potrebbe pertanto essere riletto, l'uso di questo attributo è una forma meno sicura, ma automatica, del tipo di cancellazione effettuato con `shred` di cui si è parlato in sez. 1.2.2.

[75] è cioè possibile aprire un file in scrittura soltanto se lo si fa nel cosiddetto *append mode*; per i dettagli sul funzionamento di questa modalità si può consultare sez. 5.2.1 di [GaPiL].

[76] a meno che non si faccia ricorso a meccanismi di controllo di accesso più sofisticati di quelli ordinari, come SELinux o le *capabilities*, che vanno però al di là dello scopo di questo testo.

[77] in sostanza, oltre ad impedirne la rimozione dalla directory in cui si trova è impossibile modificare sia il contenuto del file che dell'*inode*, pertanto anche tutte le altre caratteristiche del file (permessi, proprietario, ecc.) non possono essere modificate.

Capitolo 2

La shell e i comandi

2.1 L'interfaccia a linea di comando.

I sistemi Unix nascono negli anni '70, ben prima della nascita delle interfacce grafiche, quando l'unico modo di interagire con il computer era attraverso dei terminali, se non addirittura delle semplici telescriventi. Per cui, anche se oggi sono disponibili delle interfacce grafiche del tutto analoghe a quelle presenti in altri sistemi operativi nati in tempi più recenti, l'interfaccia a riga di comando resta di fondamentale importanza, dato che 30 anni di storia e migliaia di persone che ci hanno lavorato sopra per migliorarla, la hanno resa l'interfaccia utente più potente e flessibile che ci sia.

2.1.1 La filosofia progettuale

Come per il sistema, anche le caratteristiche dell'interfaccia a riga di comando derivano da alcune scelte progettuali precise. Arnold Robbins spiega molto chiaramente questa filosofia in un articolo riportato anche nella pagina *info* (vedi sez. 2.4.2) del pacchetto dei `coreutils` GNU. In sostanza la filosofia progettuale della shell e dei comandi a riga di comando si può capire facendo ricorso ad una analogia, che riprenderemo da quell'articolo.

Molte persone utilizzano un coltellino svizzero, dato che questo permette di avere in solo oggetto un discreto insieme di attrezzi diversi: coltello, forbici, cacciavite, seghetto, cavatappi. Però è molto difficile vedere un professionista usare il coltellino svizzero per fare il suo lavoro. Un professionista ha bisogno di attrezzi professionali, e un carpentiere non costruisce una casa con un coltellino svizzero, ma con tanti attrezzi, ciascuno dei quali è specializzato nello svolgere un compito specifico.

Le persone che hanno progettato l'interfaccia a riga di comando erano appunto dei professionisti, che sapevano bene che anche se fare un programma unico per tutti i compiti poteva essere attraente per l'utente finale, che deve conoscere solo quello, in pratica questo sarebbe stato difficile da scrivere, mantenere e soprattutto estendere. Per cui da professionisti pensarono ai programmi come a degli attrezzi, e piuttosto che il coltellino svizzero realizzarono l'equivalente della cassetta degli attrezzi (quella che in gergo viene chiamata "*Unix toolbox*"), con in testa un criterio fondamentale: che ciascun programma facesse una sola cosa, nel miglior modo possibile.

Questa è la caratteristica fondamentale dei programmi base di un sistema unix-like come GNU/Linux. Ogni comando è progettato per eseguire un compito preciso. Ne abbiamo incontrati già diversi nel corso della trattazione delle caratteristiche del sistema nel precedente capitolo: `ls` mostra la lista dei file, `ps` la lista dei processi, `cp` copia un file, `chmod` cambia i permessi, `man` mostra le pagine di manuale, ecc. I comandi hanno uno scopo preciso e precise funzionalità; le opzioni sono limitate e comunque specifiche allo scopo del comando, e sono descritte dettagliatamente nella relativa pagina di manuale.

Il passo successivo fu quello di costruire anche un meccanismo che permettesse di combinare insieme i vari programmi, cosicché divenisse possibile eseguire, con una opportuna combinazione, anche dei compiti che nessuno di essi era in grado di fare da solo. Questo aveva il grande vantaggio, rispetto all'approccio del programma universale, di non dover attendere che l'autore dello stesso si decidesse a programmare la funzione in più che serviva e che non era stata prevista all'inizio.

Questo è il ruolo della *shell*, cioè del programma che implementa l'interfaccia a riga di comando; è attraverso di essa che, concatenando vari comandi, si può costruire l'equivalente di una catena di montaggio, in cui il risultato di un comando viene inviato al successivo, riuscendo così a realizzare compiti complessi con grande velocità e flessibilità, e spesso a fare anche cose che gli autori dei singoli programmi non si sarebbero neanche immaginati.

La modalità tradizionale con cui si utilizza l'interfaccia a riga di comando è attraverso una sessione di lavoro interattiva eseguita su un terminale. Un tempo questo accesso avveniva con la classica procedura di collegamento al sistema sulla console (torneremo su questo in sez. 4.3.5) dove un apposito programma, `login`, una volta autenticato l'utente, mette in esecuzione la *shell*.

Oggi, con la presenza sempre più diffusa delle interfacce grafiche e la diffusione delle reti, si hanno molte altre modalità di accesso ad un terminale. Ad esempio si può avere un opportuno programma, il cosiddetto *emulatore di terminale*, che opera all'interno dell'interfaccia grafica creando un terminale virtuale a cui si accede all'interno di una finestra, oppure si può accedere via rete attraverso un programma di collegamento remoto ed anche in questo caso esistono diversi programmi e diversi protocolli di collegamento (vedremo il principale in sez. 8.3). In tutti questi casi comunque, una volta predisposta l'opportuna interfaccia di accesso, viene sempre lanciata una shell.

Si ricordi comunque che per il kernel, secondo la filosofia fondamentale di Unix illustrata in sez. 1.1.1, la shell resta un programma come tutti gli altri; essa ha però un compito fondamentale, che è quello di fornire l'interfaccia che permette di lanciare altri programmi. Inoltre è sempre la shell che permette di usufruire di tutta una serie di ulteriori funzionalità messe a disposizione dal kernel, come il controllo di sessione trattato in sez. 2.1.2.

Dato che la shell è un programma come gli altri, essa può essere realizzata in diversi modi, ed in effetti nel tempo sono state realizzate diverse shell. Anche in questo caso ci sono stati due filoni di sviluppo, il primo deriva dalla prima shell realizzata, la *Bourne shell*, chiamata così dal nome del suo creatore. La *Bourne shell* è la shell più antica e le sue funzionalità sono anche state standardizzate nello standard POSIX.2. Il secondo filone nasce da un'altra shell, che usa una sintassi leggermente diversa, con delle analogie con quella del linguaggio C, e che per questo venne chiamata *C shell*.

Ciascuno di questi due filoni ha dato vita a successive versioni di shell con funzionalità più o meno avanzate; un breve elenco di quelle più significative disponibili su GNU/Linux è il seguente:

- ***Bourne shell e derivate***.
  - **La Bourne shell**. La prima shell di Unix, in genere utilizzata semplicemente con il comando `sh`. Non viene praticamente più usata. In GNU/Linux è sostituita da `bash`. che quando viene invocata come `sh` fornisce esclusivamente le funzionalità previste dallo standard POSIX.2, disabilitando le varie estensioni di cui è dotata, o da `dash`. Sugli altri sistemi che rispettano lo standard POSIX.2, è di norma sostituita da `ksh`.
  - **La Bourne-Again Shell**. La *bash* è la shell di riferimento del progetto GNU. Il suo nome è un gioco di parole sul nome della *Bourne shell*, in sostanza una shell "*rinata*". Viene utilizzata con il comando `bash`. Incorpora molte funzionalità avanzate, come la storia dei comandi (detta *history*), l'auto-completamento dell'input sulla linea di comando (per comandi, nomi di file e qualunque altra cosa, date le opportune estensioni), editing di linea, costrutti di programmazione complessi e molto altro (praticamente di tutto, si vocifera sia anche in grado di fare il caffè).
  - **La Korn Shell** La Korn shell (dal nome dell'autore) è stata la prima ad introdurre la *history* (l'accesso ai comandi precedenti) e l'editing della linea di comando. Ha il grosso difetto che gran parte delle funzionalità avanzate non vengono attivate di default, per cui occorre un ulteriore lavoro di configurazione per utilizzarla al meglio. Viene utilizzata con il comando `ksh`. Non viene usata su GNU/Linux dato che `bash` ne ha tutte le caratteristiche; è però utile conoscerne l'esistenza dato che è facile trovarla su altri Unix.
  - **La ash**. Una shell *minimale*, realizzata in poche decine di kilobyte di codice sorgente. Viene utilizzata con il comando `ash`. Ha molti comandi integrati, occupa poca RAM e poco spazio disco, ed ha poche funzioni (ma è conforme allo standard POSIX.2). Viene usata spesso nei dischetti di installazione o recupero e nei sistemi embedded, può essere utile per sistemi dove si fa un grosso uso di script semplici perché è più veloce di `bash`.
  - **La Z shell**. Un'altra shell avanzata. Viene utilizzata con il comando `zsh`. Offre praticamente le stesse funzioni della Korn shell, ed altre funzionalità avanzate, come il completamento di comandi, file e argomenti, che però trovate anche nella `bash`.
- ***C shell e derivate***.
  - **La C shell**. Utilizza una sintassi con analogie a quella del linguaggio C. Viene utilizzata con il comando `csh`. In GNU/Linux non è disponibile essendo sostituita da `tcsh`.
  - **La tcsh**. È una evoluzione della *C shell*, alla quale aggiunge *history* e editing di linea e varie funzionalità avanzate. Viene utilizzata con il comando `tcsh`. Si trova su vari Unix proprietari, ma è poco diffusa su GNU/Linux, pur essendo disponibile.

Dato che è il principale strumento di lavoro di un amministratore professionista, la scelta della shell è spesso una questione strettamente personale. Qui parleremo però solo di `bash`, che è la shell utilizzata in praticamente tutte le distribuzioni di GNU/Linux, e probabilmente è anche la più potente e flessibile fra quelle disponibili. L'unico motivo per volerne usare un'altra infatti è

solo perché siete maggiormente pratici con quella, nel qual caso probabilmente non avete bisogno di leggere questo capitolo.

Il riferimento più immediato per il funzionamento della `bash` è la sua pagina di manuale, accessibile al solito con `man bash`. Probabilmente questa è una delle più lunghe fra tutte le pagine di manuale: quella associata alla versione 4.2 consta di ben 5459 righe, ed in effetti più che una pagina è un manuale. Per questo in seguito faremo riferimento, quando necessario, alle varie sezioni in cui essa è divisa. Per le funzionalità più avanzate esiste anche un ottimo manuale libero [AdvBash], tradotto pure in italiano.

2.1.2 Sessioni di lavoro e *job control*

Come accennato in sez. 2.1.1, l'uso dell'interfaccia a riga di comando è strettamente legato all'uso di un terminale sul quale opera una shell, il *terminale di controllo*, all'interno di quella che viene solitamente chiamata una "*sessione di lavoro*". Questo ci aveva già portato in sez. 1.3.2 a tracciare una distinzione fra i processi che avevamo classificato come interattivi, perché associati ad un terminale di controllo, visibile nell'output di `ps` alla colonna `TTY`, e quelli non interattivi, o meglio quelli che non lo sono tramite l'interfaccia a riga di comando, che non hanno nessun terminale di controllo.

Ai tempi in cui si aveva a disposizione un solo terminale (Unix è nato quando ancora i terminali erano delle telescriventi, ed anche in seguito venivano utilizzati terminali seriali come i VT100) se questo fosse stato utilizzabile da un solo processo alla volta, non si sarebbero potute sfruttare le capacità di multitasking del sistema. Per questo vennero introdotte delle funzionalità che permettessero di lanciare ed usare contemporaneamente più programmi attraverso un solo terminale, cosa che viene realizzata appunto attraverso il sistema del cosiddetto *controllo di sessione*. Oggi, con la possibilità di avere più *console* virtuali, e con l'interfaccia grafica che non fa riferimento ad un terminale, questo problema non c'è più, ma l'interfaccia è rimasta e mantiene comunque una sua utilità, ad esempio in presenza di una connessione diretta via modem in cui si ha a disposizione un solo terminale, e si devono eseguire compiti diversi.

Per spiegare le modalità in cui viene realizzata l'interazione tramite il terminale, occorre anticipare una caratteristica fondamentale dell'interfaccia a riga di comando. Tutte le volte che si lancia un programma che deve interagire con la riga di comando, questo si aspetta di avere a disposizione e già aperti tre file (torneremo su questo, e su come utilizzare queste funzionalità, in sez. 2.1.4) che convenzionalmente sono chiamati *standard input*, *standard output* e *standard error*. Dato che questi sono i primi tre file aperti e lo sono in questa sequenza, ad essi vengono sempre assegnati rispettivamente i *file descriptor* 0, 1 e 2.[1]

Convenzionalmente un programma legge i suoi dati in ingresso dal primo file descriptor, scrive i dati in uscita sul secondo, e gli eventuali errori sul terzo. Si tenga presente che è solo una convenzione, anche se seguita in maniera universale, si può tranquillamente scrivere un programma che si comporta in modo diverso, scrivendo e leggendo da un file descriptor qualunque, ma ovviamente si avranno problemi di interazione con gli altri programmi.

Quando un processo è interattivo nel senso appena illustrato, tutti e tre questi file corrispondono al terminale di controllo, e l'interfaccia dei terminali (cioè quella dei dispositivi di questo

[1] come accennato in sez. 1.2.2 il sistema mantiene anche una lista dei file aperti dai vari processi, ciascun processo a sua volta ha una lista dei file che ha aperto, il cui indice è appunto un numero chiamato *file descriptor*, per maggiori dettagli riguardo a questa problematica si può consultare sez. 5.1.1 di [GaPiL].

tipo) fa sì che in lettura il kernel faccia arrivare sullo *standard input* quanto scritto sulla tastiera e che quanto scritto sullo *standard output* venga stampato sullo schermo, o sulla finestra dentro l'interfaccia grafica, qualora si usi un emulatore di terminale, o trasmesso via rete, se si usa un terminale associato ad una connessione remota. Dato che, come accennato in sez. 1.3.1, i file aperti vengono ereditati dai processi figli e mantenuti nell'esecuzione di un nuovo programma,[2] basta che questi file siano aperti all'inizio, e lo resteranno per tutti i processi lanciati in seguito.

Per capire meglio questa infrastruttura vediamo un esempio della sua applicazione. Come accennato una delle modalità per utilizzare l'interfaccia a riga di comando è quella di collegarsi ad una *console* con la tradizionale procedura di login su terminale; nei risultati del comando `ps` mostrati a pag. 27 si può notare come sia presente il processo `getty` associato ad alcuni terminali virtuali (da `tty1` a `tty6`).

Questo è il programma che normalmente cura la procedura di login su una *console*, dando l'avvio alla conseguente sessione di lavoro, quasi tutte le distribuzioni infatti attivano nella procedura di avvio (tratteremo l'argomento in sez. 5.3.5) sei *console* virtuali a cui si accede con la combinazione di tasti `Alt-FN`, dove "`N`" è il numero della rispettiva *console*, cosa che consente di avere diverse sessioni di lavoro usando un solo monitor ed una sola tastiera.

In questo caso è `getty` che si cura dell'apertura iniziale dei tre file di *standard input*, *standard output* e *standard error* facendo riferimento al terminale che gli è stato indicato, nel caso una delle *console* virtuali, corrispondenti ai terminali da `/dev/tty1` a `/dev/tty6`, che diventerà il terminale di controllo di tutta la seguente sessione di lavoro. Poi il programma stampa un messaggio di benvenuto e la linea "`login: `" e si pone in attesa che un utente scriva sul terminale il suo "*username*" (il cosiddetto *nome utente* o *nome di login*), che come accennato è il nome con cui un utente si identifica presso il sistema (torneremo sull'argomento in sez. 4.3.1).

Una volta che questo sia stato scritto, `getty` esegue direttamente il programma `login` che si cura di chiedere la password ed effettuare l'autenticazione[3]. Se questa ha successo è `login` che si incarica di impostare il valore del *session ID* per il processo in corso, cambiare gli identificativi ad esso associati (cioè l'*user ID* ed il *group ID* visti in sez. 1.4.1) per farli corrispondere a quelli dell'utente che si è collegato, ed infine lanciare quella che si chiama una "*shell di login*".

In questo caso il programma invocato è sempre lo stesso, ma dato che come vedremo più avanti una shell può essere utilizzata con modalità diverse (all'interno di uno script non interattivo, o come *sub-shell* all'interno di una sessione) esistono delle opportune opzioni che consentono di indicare la modalità di utilizzo: quella che indica una *shell di login* è una di queste, (torneremo sull'argomento in sez. 2.1.6). In tutta questa procedura i tre file di *standard input*, *standard output* e *standard error* resteranno aperti ed associati al terminale di controllo da cui si è partiti, che è quello su cui verrà utilizzata la shell.

Una volta completata la procedura di collegamento avremo a disposizione una shell con un certo valore del *session ID*, e dato che anche questa è una proprietà che viene conservata quando si creano processi figli e si lanciano altri comandi, si avrà come conseguenza che anche tutti i processi lanciati da detta shell saranno identificati dallo stesso *session ID*. Tutto ciò vale anche per tutte le altre modalità con cui si può iniziare una sessione di lavoro su un terminale, che sia attraverso un emulatore ad interfaccia grafica o un programma di connessione remota.

[2] l'argomento è in realtà molto complesso e qui stiamo semplificando brutalmente, per una trattazione dettagliata si consultino sez. 3.1 e sez. 5.1 di [GaPiL].

[3] in questo caso si tratta proprio dell'invocazione diretta del comando senza passare attraverso un nuovo processo, questo significa che il PID del processo resterà lo stesso.

Come accennato in sez. 1.3.2 le informazioni relative alla sessione possono essere visualizzate con l'opzione `-j` di `ps`, ad esempio lanciando il comando dal terminale in cui si stava scrivendo questo testo abbiamo ottenuto il seguente risultato:

```
piccardi@anarres:~/truedoc/corso$ ps -je f

  PID  PGID   SID TTY      STAT   TIME COMMAND
    1     0     0 ?        S      0:02 init [2]
...
 1820  1820  1820 tty1     Ss     0:00 /bin/login --
 2112  2112  1820 tty1     S      0:00  \_ -bash
18911 18911  1820 tty1     S+     0:00      \_ top
...
 2882  2214  2214 ?        Sl     0:20 gnome-terminal
 2886  2214  2214 ?        S      0:00  \_ gnome-pty-helper
15894 15894 15894 pts/3    Ss     0:00  \_ bash
16065 16065 15894 pts/3    S      0:33  |   \_ emacs struttura.tex
16073 16073 15894 pts/3    Sl     0:29  |   \_ evince corso.pdf
18552 18552 15894 pts/3    R+     0:00  |   \_ ps -je f
17030 17030 17030 pts/0    Ss     0:00  \_ bash
20548 20548 17030 pts/0    S+     0:00      \_ watch ls -lR / | wc -l
20549 20548 17030 pts/0    S+     0:00          \_ watch ls -lR / | wc -l
20550 20548 17030 pts/0    S+     0:00              \_ sh -c ls -lR / | wc -l
20551 20548 17030 pts/0    R+     0:02                  \_ ls -lR /
20552 20548 17030 pts/0    S+     0:00                  \_ wc -l
...
```

In questo caso la shell su cui si sta lavorando è stata lanciata all'interno di una sessione associata al terminale virtuale `pts/3`, creata da `gnome-terminal` all'interno dell'interfaccia grafica. Tutti i programmi che si sono lanciati da questa shell (`evince`, `emacs` e lo stesso `ps`) hanno lo stesso valore nella colonna `SID` dato che appartengono alla stessa sessione di lavoro. Lo stesso dicasi per l'altra sessione di lavoro in una diversa finestra di `gnome-terminal` dove viene eseguito `watch`, e per la sessione di lavoro sulla *console* `tty1` lanciata da `getty` che sta eseguendo `top`.

Il risultato di `ps` mostra anche una seconda caratteristica delle sessioni di lavoro, associata a quello che viene chiamato *job control*: quando si lancia una linea di comando che comporta l'avvio di altri processi, come quella con `watch`, tutti questi processi (nel caso `ls`, `wc` e la sub-shell che li esegue) vengono assegnati allo stesso *process group*, e pertanto hanno lo stesso valore del relativo identificativo nella colonna `PGID`.

La modalità più comune di uso della shell è quella in cui si lancia un comando, si aspetta che questo completi le operazioni, e poi se ne lancia un altro, e così via. Quello che succede in questo caso è che per ogni comando la shell crea un nuovo processo e si ferma in attesa che questo completi le sue operazioni. Alla conclusione del programma la shell riprende il possesso del terminale e consente di lanciare un altro comando.[4] In genere si lavora in questo modo perché tutte le volte che si esegue un programma interattivo questo deve avere a disposizione il terminale di controllo per ricevere i dati in ingresso ed inviare i dati in uscita.

Ovviamente se fosse possibile lavorare solo in questo modo si perderebbe la possibilità di eseguire contemporaneamente più comandi e di sfruttare le capacità multitasking del sistema. Se però si osserva il precedente risultato di `ps` si potrà notare come tutti i comandi citati (`evince`

[4] se si ricorda quanto detto in sez. 1.3.2 riguardo alla gestione della conclusione dei processi figli, essendo la shell il processo padre dei comandi che mette in esecuzione, essa viene sempre notificata anche della loro conclusione.

ed `emacs`), pur facendo riferimento allo stesso terminale, vengono eseguiti insieme: essi infatti sono stati lanciati come suol dirsi in *background*, in una modalità cioè in cui essi non sono agganciati al terminale.

Questa è un'altra funzionalità della shell che consente, quando si termina una linea di comando con il carattere "`&`", di mandare in esecuzione i processi sganciati dal terminale, così che questo resti a disposizione della shell stessa per eseguire altri comandi. Se, come accade per i citati `evince` ed `emacs` con cui si interagisce attraverso l'interfaccia grafica, il programma non ha bisogno di accedere al terminale per funzionare, esso potrà continuare ad eseguire le sue operazioni restando appunto "*sullo sfondo*".

Altrimenti al primo tentativo di accesso il programma riceverà dal kernel un opportuno segnale in modo da bloccarne l'esecuzione,[5] anzi in realtà il segnale verrà inviato a tutti i processi che fanno parte dello stesso *process group*, perché, come illustrato nei risultati di `ps` per quanto riguarda la sessione in cui viene eseguito `watch`, e come vedremo meglio anche in sez. 2.1.4, si possono eseguire più processi su una singola riga di comando e se il terminale non è disponibile dovranno essere fermati tutti.

Vediamo allora un esempio di questo comportamento, se si lancia in *background* un editor testuale come `jed`, otterremo che l'esecuzione del comando viene immediatamente fermata, dato che ovviamente un editor necessita di interagire con il terminale per poter essere utilizzato e la shell stamperà un avviso per avvisarci dell'avvenuto blocco del programma;[6] avremo cioè qualcosa del tipo:

```
piccardi@anarres:~/truedoc/corso$ jed prova &
[3] 22913

[3]+  Stopped                 jed prova
```

Tutte le volte che, come nell'esempio, si manda un processo in *background*, la shell stampa sul terminale un valore numerico fra parentesi quadre, detto *job ID* (dato che in gergo si suole chiamare *job* un processo che lavora in *background*) seguito dal PID del processo. La shell infatti identifica i processi attivi all'interno di una stessa sessione di lavoro assegnando loro un numero progressivo, il *job ID*, che poi può essere utilizzato come riferimento nei vari comandi di gestione del *job control*. L'esempio ci mostra anche come la shell provveda a stampare una notifica sul terminale tutte le volte che un processo viene bloccato, utilizzando di nuovo il *job ID*.

Oltre a lanciare un processo in *background* ci sono casi in cui può esser necessario fermare temporaneamente un comando in esecuzione, per eseguire un controllo o un qualche altro compito più urgente. Abbiamo visto in sez. 1.3.3 che è possibile fermare l'esecuzione di un processo con l'invio dei segnali `SIGSTOP` o `SIGTSTP`, il sistema del controllo di sessione semplifica questa operazione permettendo di inviare `SIGTSTP` ai processi che sono attivi su un terminale, tramite la combinazione di tasti `C-z`, cioè l'uso del tasto "*control*" insieme alla lettera "`z`" (la notazione è formalizzata in sez. 2.3.2).

Di nuovo la shell si accorgerà che il processo è stato fermato stampando, come nell'esempio precedente, un messaggio di notifica prima di rimetterci a disposizione il terminale. Oltre a `SIGTSTP` quando un processo è associato ad un terminale di controllo gli si possono mandare altri

[5] i segnali sono rispettivamente `SIGTTIN` se l'accesso è in lettura e `SIGTTOU` se l'accesso in scrittura e l'azione di default di entrambi è quella di sospendere l'esecuzione del processo.

[6] il sistema prevede anche che un processo padre possa essere notificato se uno dei suoi figli viene fermato, per i dettagli si consulti sez. 3.1.5 di [GaPiL].

segnali con delle opportune combinazioni di tasti: `C-c` invia un `SIGINT` e `C-\` invia un `SIGQUIT`. Un elenco delle principali combinazioni di tasti supportate dall'interfaccia dei terminali è riportata in tab. 2.1.

Combinazione	**Significato**
`C-z`	sospende l'esecuzione del comando corrente con `SIGTSTP`.
`C-c`	interrompe l'esecuzione del comando corrente con `SIGINT`.
`C-\`	interrompe l'esecuzione del comando corrente con `SIGQUIT`.
`C-d`	invia il carattere di *end-of-file*, che chiude lo *standard input*.
`C-s`	blocca le operazioni di I/O sul terminale.[7]
`C-q`	riabilita le operazioni di I/O sul terminale.

Tabella 2.1: Le scorciatoie da tastiera dell'interfaccia dei terminali.

L'elenco dei processi in *background* o bloccati sul terminale può essere stampato con il comando `jobs`, che mostra la lista e lo stato di ciascuno di essi. Si tenga presente che questo, come i seguenti `bg`, `fg` e `wait`, è un comando interno della shell, e non corrisponde a nessun eseguibile su disco, (vedremo la differenza in sez. 2.1.3). Di default il comando stampa l'elenco ordinandolo per *job ID*, ma gli si possono fare stampare anche i PID con l'opzione `-l`. L'elenco riporta il *job ID* in prima colonna, seguito dallo stato e dal comando eseguito, nella forma:

```
piccardi@anarres:~/truedoc/corso$ jobs
[1]   Running                 emacs struttura.tex &
[2]-  Running                 evince corso.pdf &
[3]+  Stopped                 jed prova
```

dove il carattere "`+`" indica il cosiddetto *job corrente*, cioè il processo su cui si è operato per ultimo, mentre il carattere "`-`" indica il *job* precedente (per maggiori dettagli consultare il manuale della `bash` alla sezione "`JOB CONTROL`").

Quando un programma è stato fermato con `C-z` la shell ci consente di farlo ripartire in *background* con il comando `bg`, ovviamente se questo dovesse cercare di eseguire un accesso a terminale si bloccherebbe immediatamente, per questo è inutile usare `bg` su un comando che si è fermato per questo motivo: si fermerà di nuovo. Un processo può essere riassociato al terminale, cioè essere rimesso, come suol dirsi, in *foreground*, con il comando `fg`, se il processo era stato fermato in attesa di poter disporre del terminale in questo modo viene anche fatto ripartire.

Se non si specifica nulla sia `bg` che `fg` che tutti gli altri comandi del controllo di sessione agiscono sul *job corrente*. Qualora si voglia invece agire su uno specifico *job* si può specificare come parametro sia il testo della riga di comando, che il numero di *job ID* stampato da `jobs` e questo può essere indicato sia direttamente che preceduto da un carattere "`%`". Quest'ultima sintassi può essere usata anche con il comando `kill`, se questo è realizzato come comando interno della shell come avviene nel caso di `bash`, per mandare i segnali ad un processo identificato per *job ID* piuttosto che per PID.

Se si vuole aspettare la fine di uno specifico comando che si è mandato in *background* si può usare il comando `wait`, passandogli di nuovo il *job ID* come argomento. Se non si specifica

[7] vengono cioè bloccate le operazioni di lettura e scrittura sul terminale, con la conseguenza di bloccare anche l'accesso alla riga di comando da parte della shell, la cui esecuzione si ferma in attesa che questo venga riabilitato; quanto si scrive sul terminale viene comunque ricevuto e memorizzato ma non ha nessun effetto, la shell infatti non lo riceverà fintanto che non si riabilitano le operazioni di I/O.

nulla il comando aspetta la fine di tutti i processi in *background*. Questo comando può essere utile negli script di shell (vedi sez. 2.1.5) quando si è lanciato un comando in *background* per poter eseguire nel frattempo altre operazioni, compiute le quali si vuole aspettare che il comando iniziale finisca.

Una volta che si voglia terminare una sessione ed uscire dalla shell si può usare il comando `exit` o la combinazione di tasti `C-d` che, come riportato in tab. 2.1, corrisponde ad inviare un carattere di *end-of-file* sul terminale, cosa che segnala alla shell che lo si vuole chiudere e con questo terminare la sessione di lavoro.

Se si cerca di uscire dalla shell e chiudere la sessione ma ci sono *job* in *background* o bloccati, la shell stamperà un avviso senza uscire, così da poter provvedere a risolvere la situazione. Se però si ripete immediatamente la richiesta di chiusura, questa verrà onorata e la shell verrà terminata insieme ai processi pendenti. Il comando `exit` prende anche un argomento opzionale che costituisce lo stato di uscita (torneremo su questo in sez. 2.1.5) e viene usato principalmente per uscire all'interno degli script .

Infine ci si può chiedere cosa succede se si interrompe la connessione ad un terminale, ad esempio se si interrompe una connessione remota, su cui sono in esecuzione dei programmi (in *background* o meno). In tal caso il kernel provvede ad inviare il segnale `SIGHUP` a tutti i processi che hanno quello come terminale di controllo, l'effetto di default di questo segnale è causare la terminazione del processo, onde evitare continui ad essere eseguito senza poter riportare nessun risultato. Il nome del segnale deriva dall'inglese "*hungup*" (*impiccato*) in quanto in casi come questo un processo resterebbe *appeso* nel vuoto senza poter avere nessuna interazione esterna.[8] È per questo motivo che se si esegue un programma da un emulatore di terminale sull'interfaccia grafica (ad esempio si lancia un editor) e poi si chiude il terminale, viene chiuso anche il programma che si è lanciato.

Esistono comunque alcune situazioni in cui questo comportamento non è desiderato, ad esempio se si vuole lanciare su una macchina remota un programma che non necessita di utilizzare il terminale, come un programma di elaborazione dati. Per evitare la terminazione in caso di disconnessione o di uscita dalla shell si può anteporre `nohup` alla riga di comando che si vuole eseguire, che a questo punto verrà eseguita senza terminale di controllo e non terminerà una volta che ci si scolleghi da esso.

In questo caso, dato che non si vuole neanche che il programma si fermi in caso di accesso al terminale, qualora esso generi dell'output questo verrà scritto in coda al file `nohup.out`, nella directory in cui si è lanciato il programma. Se questo file non fosse accessibile verrebbe usato al suo posto un file `nohup.out` nella *home* dell'utente, ed in caso di inaccessibilità anche di quest'ultimo si avrebbe un errore.

La `bash` consente anche, qualora si sia già lanciato un programma, di utilizzare il comando `disown` per disassociare un processo in *background* dalla tabella dei *job*, ed evitare così che gli venga inviato il segnale di `SIGHUP` all'uscita della shell, si può comunque mantenere il programma nella lista dei *job* e disabilitare solo l'invio di `SIGHUP` usando l'opzione `-h`. Se non si specifica nulla il comando si applica al *job corrente*, altrimenti può specificare uno o più *job ID*, o l'opzione `-r`

[8]abbiamo accennato in sez. 1.3.3 come questo segnale venga utilizzato anche per dire ai demoni di sistema di rileggere i propri file di configurazione; questo non è in contrasto con quanto appena detto, in quanto detti programmi sono caratterizzati dal non avere un terminale di controllo, per cui `SIGHUP` può essere riutilizzato con un significato diverso.

per indicare i programmi attualmente in esecuzione o l'opzione `-a` per indicare tutti i programmi della sessione di lavoro.

Entrambe queste soluzioni però rendono impossibile riprendere in mano una sessione avviata e non funzionano se il comando per un qualche motivo richiede dei dati in ingresso. Per ovviare a questo inconveniente e rendere possibile interrompere e riprendere una sessione di lavoro in qualunque momento, si può invece usare il comando `screen`, che consente di creare un numero arbitrario di sessioni indipendenti gestite all'interno del comando stesso. In questo modo è possibile distaccarsi da una di queste sessioni lasciandovi in esecuzione i propri programmi, e ricollegarsi in un secondo momento (si può ad esempio passare da un collegamento in locale ad uno remoto via SSH).

Se lanciato senza argomenti `screen` apre una nuova sessione, chiedendo di confermarne la creazione con un invio, e vi si *aggancia*. Il programma simula un terminale virtuale il cui contenuto viene riportato sul terminale da cui lo si è invocato. Si può anche specificare come argomento una riga di comando che verrà eseguita all'interno della sessione stessa. All'interno della sessione il programma intercetta la combinazione di tasti `ctrl-a`, per eseguire le proprie operazioni. Ad esempio con `ctrl-a d` ci si *distacca* dalla sessione di `screen` tornando nella sessione di partenza, con `ctrl-a h` si salva sul file `hardcopy.`*n* una copia della schermata, con con `ctrl-a a` si passa un `ctrl-a` al terminale sottostante. Con opportune combinazioni (per le quali si rimanda alle pagine di manuale) è possibile anche creare delle ulteriori *finestre* interne, associate ad altrettante sessioni e terminali, in cui eseguire altri programmi in maniera totalmente indipendente gli uni dagli altri e passare dall'una all'altra.

Quanto eseguito all'interno di una sessione di `screen` userà il terminale virtuale ad essa associato, sia che questo sia associato ad un terminale da cui si osserva la sessione, sia che sia *distaccato*. Si tenga presente che tratta comunque di un terminale elementare (un VT100), con una limitata memoria per la visualizzazione dello *standard output*, per cui in genere restano disponibili solo le ultime righe dell'output del comando. Una volta distaccatisi si potrà riagganciarsi ad una sessione avviata in precedenza invocando il comando da un qualunque altro terminale con l'opzione `-r`.

Opzione	Significato
`-d` *[sess]*	forza la disconnessione di una sessione agganciata ad un'altra istanza del comando, si può abbinare a `-r` per forzare la connessione e a `-m` per creare una sessione disconnessa.
`-h`	imposta il numero di righe di terminale da ricordare nello scorrimento all'indietro del terminale virtuale interno.
`-r` *[sess]*	si aggancia ad una sessione (che deve essere distaccata) se ve ne sono disponibili più di una si deve indicare quale.
`-R`	si connette alla sessione distaccata più recente, creandone una nuova se nessuna è disponibile.
`-S` *sess*	imposta un nome specifico per la sessione, sovrascrive il default di *pid*.*tty*.`host`.
`-x`	forza la connessione anche se la sessione è gestita da un'altra istanza del comando, condividendo il terminale virtuale.
`-X`	invia il comando passato come argomento ad una sessione (se ve ne sono disponibili più di una si deve specificare quale con `-S`).

Tabella 2.2: Principali opzione del comando `screen`.

Se è disponibile una sola sessione di lavoro si verrà immediatamente riagganciati alla stessa, mentre se ve ne sono più di una il comando fallirà stampando la lista delle stesse, nella forma `pid.tty.host` che consente di identificarle univocamente, e si potrà selezionare quella voluta passando la relativa stringa identificativa come argomento per `-r`. Ci si può collegare solo alle sessioni create con il proprio utente, solo l'amministratore è in grado di agganciarsi a sessioni di altri utenti, usando una stringa identificativa nella forma `user/pid.tty.host`.

Si tenga presente comunque che di default `screen` non consente di agganciarsi ad una sessione ancora agganciata ad un'altra istanza del programma; in questo caso se ne può forzare la disconnessione specificando l'ulteriore opzione `-d` che può essere usata anche da sola per forzare il distacco. Ci si può inoltre collegare anche senza forzare la disconnessione con l'opzione `-x`, in tal caso le operazioni saranno effettuabili da una qualunque delle istanze collegate, ottenendo una sorta di condivisione dello schermo. Le altre principali opzioni di `screen` sono riportate in tab. 2.2, per la documentazione ed i dettagli si rimanda al solito alla pagina di manuale.

2.1.3 Sintassi e funzionalità di base della riga di comando

Finora si è parlato di riga di comando e del fatto che la shell consente all'utente di lanciare i vari programmi che vuole utilizzare, sorvolando però sui dettagli delle modalità in cui questo viene fatto. In realtà la shell è un programma molto complesso, e quello di interprete della riga di comando è soltanto uno dei modi in cui può essere usata (torneremo su questo in sez. 2.1.5), inoltre anche in questo caso (quando cioè si ha a che fare con quella che si suole chiamare una *shell interattiva*) ci si è limitati a mostrare esempi elementari.

Per comprendere le reali capacità della shell e trattare la sintassi della riga di comando occorre prima approfondire le modalità con cui la shell legge una riga dalla tastiera, riconosce qual è il programma che volete mettere in esecuzione, ricostruisce i vari argomenti da passare al programma stesso, e poi lo esegue. L'invocazione generica di un qualunque comando è nella forma:[9]

```
comando -o -v valore --altra-opzione argomento1 --riopzione=valore arg2
```

da scrivere sulla stessa riga e terminare con la pressione del tasto di invio.

La pressione del tasto di invio su un terminale infatti serve a "*scaricare*" la linea appena scritta in modo che questa possa essere letta in ingresso dal programma che usa il terminale (nel nostro caso la shell). Se si vuole scrivere un comando su più righe andando a capo occorrerà proteggere la pressione del tasto di invio; questo si fa utilizzando il carattere "\" come ultimo carattere della riga, vedremo più avanti che "\" può essere usato nello stesso modo con gli altri caratteri.

La sintassi elementare della shell prevede che si scriva all'inizio della riga (eventuali spazi antistanti verranno comunque ignorati) il nome del comando da eseguire, seguito da eventuali opzioni ed argomenti. Si tenga presente che la shell, per identificare il comando e separarlo da opzioni e argomenti, e separare questi ultimi fra di loro, usa caratteri vuoti come lo spazio o il tabulatore. La presenza cioè di uno o più spazi o tabulatori dice che si sta passando dal nome del comando ad un argomento o opzione, o da un argomento/opzione al successivo. Lo stesso dicasi qualora si vada a capo su una nuova riga con l'uso del "\".

[9] in realtà anche in questo caso ci stiamo limitando ad una invocazione elementare, vedremo in seguito come l'uso di una serie di caratteri di controllo consente di modificare il comportamento della shell.

Per questo motivo il nome di un comando non contiene spazi, inoltre la stragrande maggioranza dei comandi usa nomi scritti esclusivamente in lettere minuscole, si ricordi infatti che un sistema unix-like è case-sensitive, fa differenza cioè fra maiuscole e minuscole. I soli caratteri non alfabetici utilizzati nei nomi dei comandi, in genere assai poco, sono il "-" e il "_" ed eventualmente i numeri. Gran parte degli altri caratteri infatti, come vedremo fra breve, vengono interpretati dalla shell assumendo significati speciali, e pertanto di nuovo non possono essere inseriti nel nome del comando, almeno non senza prendere adeguate precauzioni, analoghe a quelle che vedremo più avanti per gli argomenti, il che poi fa sì che si finisca per usare nomi con semplici caratteri alfanumerici.

La differenza fra opzioni e argomenti è che le prime attivano funzionalità o modalità di operazione specifiche del comando, mentre i secondi indicano gli oggetti, solitamente dei file, su cui questo opera. In realtà il meccanismo è un po' più complesso, e può essere compreso nei dettagli solo con un po' di nozioni di programmazione.[10] Quello che in effetti fa la shell è identificare il programma da usare sulla base del nome del comando, per poi spezzare l'intera riga in una lista di stringhe che verranno passate al suddetto programma come argomenti iniziali.

Questa scansione viene eseguita su tutta la riga, utilizzando gli spazi vuoti come separatori e senza distinguere fra nome del comando, opzioni e argomenti; tutto il contenuto sarà suddiviso in una lista di stringhe, che poi il programma dovrà analizzare al suo interno. Si tenga presente che opzioni ed argomenti possono essere separati da un numero arbitrario di caratteri di spaziatura, che saranno scartati. Vedremo più avanti come questo finisca con l'avere diverse conseguenze non del tutto intuitive.

Per gestire gli argomenti e le opzioni passate dalla linea di comando la gran parte dei programmi utilizzano una serie di funzioni fornite dalla libreria fondamentale del sistema, la *GNU C library* che abbiamo già incontrato in fig. 1.1.[11] Per i programmi che usano queste funzionalità questo comporta la presenza di un formato della riga di comando abbastanza uniforme, mostrato appunto nell'esempio precedente, con la possibilità di distinguere fra opzioni e argomenti, e di specificare questi in un ordine qualunque. Si tenga presente che questo non è sempre vero, dato che non tutti i programmi fanno ricorso a queste funzionalità. Per questo motivo troveremo in seguito diverse eccezioni, ad esempio in cui l'ordine conta, o in cui le opzioni vanno specificate prima degli argomenti.

Nel comportamento comune le opzioni sono sempre costituite da un carattere "-" seguito da una lettera ed in rari casi anche da numeri o caratteri non alfabetici, a cui può seguire (separato o meno da uno spazio) un eventuale valore passato come parametro, ma non tutte le opzioni prevedono parametri, nel qual caso va specificata solo la lettera dell'opzione. Se si specificano più opzioni è normalmente supportata anche la possibilità di indicarle con più lettere scritte di fila dopo un "-" iniziale.

Una forma alternativa per le opzioni, in uso principalmente con i programmi del progetto GNU che utilizzano le funzionalità della citata *GNU C library* e anch'essa mostrata nell'esempio precedente, è quella in cui l'opzione è scritta in forma *estesa*, e allora viene introdotta da un "--".

[10] per i curiosi questo argomento è trattato in dettaglio nella sezione 2.3 di [GaPiL].

[11] la funzionalità si chiama *getopt*, e permette l'uso di una sintassi standard per le opzioni, la scansione automatica delle stesse indipendentemente dall'ordine in cui le si specificano, la possibilità di identificare opzioni con parametri o senza, la capacità di passare al programma, una volta terminata la scansione della lista di stringhe ottenuta dalla shell, solo quelle contenenti gli argomenti restanti; per una trattazione più dettagliata si può fare riferimento alla sez. 2.3.2 di [GaPiL].

In questo caso, qualora essa necessiti di un parametro, il relativo valore deve essere assegnato con l'uso di un "=", ma anche questa è una convenzione, non è detto sia sempre supportata, ed in alcuni casi si può scrivere direttamente il parametro senza l'assegnazione. Torneremo su questo, e su alcuni aspetti che riguardano gli effetti non immediatamente evidenti dell'uso del carattere "-" nella sintassi della riga di comando, in sez. 2.2.1.

Per capire meglio questo comportamento facciamo un esempio; si consideri di aver scritto su un terminale una riga di comando contenente qualcosa del tipo:

```
rm -f pippo pluto paperino
```

la shell capirà che si intende invocare il comando `rm` e, individuato su disco il file che contiene il relativo programma, lo lancerà passandogli come argomenti le cinque stringhe presenti nella riga precedente. La prima stringa, `rm`, contiene il nome del programma e di norma viene ignorata, le restanti invece verranno analizzate dal programma; così `-f` verrà identificata come opzione mentre, non necessitando questa di nessun parametro, `pippo`, `pluto` e `paperino` verranno considerati come argomenti, che nel caso indicano i file da cancellare.

In realtà alcuni programmi usano anche il nome stesso con cui vengono invocati per modificare il loro comportamento, un esempio classico di questo comportamento è `gzip`, un compressore di file su cui torneremo in sez. 2.2.4, che può essere invocato anche come `gunzip` (in genere entrambi i comandi sono presenti come *hard link* allo stesso file eseguibile) nel qual caso decomprimerà invece di comprimere.

Si tenga conto poi che benché la filosofia di Unix sia quella di utilizzare un apposito comando per effettuare ciascun compito specifico, e che il caso più comune sia quello in cui la shell interpreta la riga di comando per lanciare il programma che le avete chiesto, esistono alcune funzionalità che essa fornisce direttamente tramite quelli che vengono detti *comandi interni* (o *built-in*). Se questo è il caso la shell si accorgerà che non è necessario eseguire nessun programma esterno, ed opererà direttamente sulla base di quanto dite sulla linea di comando. Di norma i comandi interni sono quelli che servono a eseguire impostazioni relative al funzionamento stesso della shell, o a impostare proprietà generali che essa deve trasmettere ai programmi che lancia.

Un esempio classico di comando interno è `cd` che viene usato per modificare la directory di lavoro corrente della shell. Un tale comando non può essere eseguito con un programma esterno, dato che nella sua esecuzione come processo separato quest'ultimo potrebbe solo cambiare la sua directory di lavoro, ma non quella della shell che lo ha lanciato. Questo perché, come accennato in sez. 1.2.2, la directory di lavoro è una delle caratteristiche che ciascun processo si porta dietro e quella iniziale è *ereditata* dal processo padre; per questo se la si cambia nella shell tutti i comandi successivi la vedranno, ma non viceversa. Altri esempi di comandi interni sono `umask` (visto in sez. 1.4.3), ed i comandi del controllo di sessione di sez. 2.1.2; ne incontreremo degli altri più avanti.

Si ricordi infine che quanto detto attiene all'uso più elementare della sintassi usata dalla shell, quello in cui si esegue un comando scrivendone per esteso il nome con le relative opzioni ed argomenti. Vedremo più avanti come sulla linea di comando si possano effettuare operazioni anche molto complesse attraverso l'uso di opportuni caratteri di controllo.

L'esempio precedente con `rm` ci mostra come la shell, oltre a quello generico di lanciare programmi, già esegua di per sé anche molti altri compiti. Il primo che prenderemo in esame è quello della visualizzazione del *prompt*, cioè di quella scritta che compare sulla sinistra della

linea di comando, a fianco della quale, se non si è scritto nulla, lampeggia il cursore, e che serve ad avvisarvi che la shell è in attesa di ricevere dei comandi da eseguire.

Nel caso della `bash` il *prompt* è completamente personalizzabile, e può contenere diverse informazioni. Quello che viene stampato come *prompt* è stabilito dalla variabile di shell `PS1`,[12] (tratteremo delle variabili di shell a breve) nel cui contenuto, oltre ai normali caratteri, che saranno stampati inalterati, si possono inserire una serie di caratteri di controllo (i principali dei quali sono riportati in tab. 2.3) che verranno automaticamente *espansi* in un determinato valore (come data, utente, stazione, ecc.). L'elenco completo è disponibile nella pagina di manuale, alla sezione "`PROMPTING`".

Opzione	**Significato**
`\d`	la data in formato tipo: `Tue May 26`.
`\h`	il nome della stazione.
`\u`	lo *username* dell'utente.
`\w`	il percorso completo della directory di lavoro.
`\W`	il nome della directory di lavoro.
`\$`	un "`#`" per l'amministratore, un "`$`" per gli altri.
`\!`	la posizione nella history del comando.
`\t`	il tempo corrente in formato HH:MM:SS (24h).
`\T`	il tempo corrente in formato HH:MM:SS (12h).

Tabella 2.3: Principali caratteri di controllo usati nella visualizzazione del prompt della shell.

Due esempi possibili per il prompt sono quello usato da Debian, che prevede per `PS1` un valore pari a "`\u@\h:\w\$`", e che produce un prompt del tipo:

```
piccardi@anarres:~/truedoc/corso$
```

o quello di RedHat, che usa un valore di "`[\u@\h \W]\$`", e produce un prompt del tipo:

```
[root@gont piccardi]#
```

La scelta del prompt dipende dai gusti e dall'uso: se ci si trova a lavorare contemporaneamente con diversi utenti, su computer diversi, allora sarà utile sapere con quale utente e su quale macchina si sta operando in un certo terminale; altrimenti si può farne a meno e risparmiare spazio sulla linea di comando con un valore di `PS1` come "`\$`".

Al di là dell'estetica del prompt il compito principale della shell è quello di mettere a disposizione dell'utente un'interfaccia che gli consenta di utilizzare al meglio il sistema; per questo essa fornisce una serie di funzionalità che permettono di rendere più efficace l'uso dei comandi. Una delle funzionalità fondamentali è quella delle *variabili di shell*; la shell permette cioè di definire e modificare delle variabili a cui si può fare riferimento in un secondo tempo. Per farlo la sintassi elementare della riga di comando vista in precedenza cambia e si deve utilizzare una riga contenente un'assegnazione nella forma:

```
VARIABILE=valore
```

[12] in realtà in tutto le variabili che controllano l'aspetto del prompt sono 4, ed oltre `PS1` ci sono anche `PS2`, `PS3` e `PS4` che vengono utilizzate in altri casi (ad esempio `PS2` viene usata come segno di continuazione di un comando quando si va a capo con l'uso di "\" a fine riga); la descrizione dettagliata del loro impiego si trova nella pagina di manuale, nella sezione "`Shell Variables`".

Se la variabile non esiste viene creata e assegnata, altrimenti viene reimpostata al valore specificato. Si noti che non ci sono spazi a separare il nome della variabile, l'operatore "=" ed il valore. In presenza di spazi infatti la shell interpreterebbe il tutto come il nome di un comando, e probabilmente riporterebbe un errore non trovando un programma corrispondente. Si tenga presente comunque che la shell supporta anche la possibilità di eseguire un comando una volta effettuata l'assegnazione di una variabile, nel qual caso questo dovrà essere specificato, separato da uno spazio, nel seguito della riga di comando.

Per convenzione le variabili di shell hanno nomi scritti con lettere maiuscole, ma questa è appunto solo una convenzione, dovuta al fatto che così è più difficile confonderle con i comandi; in realtà è possibile usare qualunque tipo di lettera e, eccezion fatta per il primo carattere, anche qualunque numero ed il carattere di sottolineatura "_".

Una volta definita una variabile di shell è possibile richiamarne il valore sulla riga di comando anteponendo al nome della variabile il carattere "$"; così se ad esempio si è inserito nella variabile MAIL il nome del file in cui si trova la propria posta elettronica, si potrà verificarne dimensioni e permessi con un comando del tipo:

```
piccardi@anarres:~$ ls -l $MAIL
-rw-rw----    1 piccardi mail        4136 Aug 25 17:30 /var/mail/piccardi
```

Questo perché la shell, quando incontra il carattere "$" seguito dal nome di una variabile, esegue quella che viene comunemente chiamata una "*variable expansion*",[13] sostituisce cioè il riferimento alla variabile (nel caso $MAIL) con il valore della stessa (nel caso /var/mail/piccardi), passando il risultato, come argomento o come opzione a seconda di dove lo si è scritto, al comando che viene eseguito.

In realtà la *variable expansion* è un procedimento molto più complesso rispetto alla descrizione grossolanamente semplificata che si è appena fatta, e senza comprenderne i dettagli si rischia di trovarsi di fronte a dei risultati inaspettati. Tutte le volte che nella scansione della riga di comando la shell incontra il carattere "$" seguito da una lettera, interpreta il tutto come l'inizio del riferimento ad una variabile, ed esegue la sostituzione con il relativo valore.

Tutto questo può avere alcuni effetti non del tutto intuitivi: il primo è che la sostituzione avviene comunque, indipendentemente dal fatto che sia stata definita o meno una variabile che corrisponde a quel nome, anche se si può forzare la stampa di un errore in questo tipo di situazione, modificando con set -u il comportamento di default della shell (vedi le opzioni di tab. 2.12).

Questo significa che normalmente anche se una variabile non esiste la sua *variable expansion* avviene lo stesso, solo che non essendoci un valore, al posto del riferimento non viene messo nulla, il che comporta che se si sbaglia a scrivere il nome di una variabile non si avrà nessun errore specifico, se non quelli dovuti al fatto che sulla riga di comando non compare più un eventuale parametro o argomento che si era espresso tramite di essa. Si ricordi infatti che prima la shell effettua la *variable expansion* e poi analizza la linea di comando che ne deriva per creare la lista di argomenti ed opzioni da passare al programma. Questo in certi casi può portare a risultati inaspettati perché l'eventuale errore deriva dalla mancanza di un argomento e non dal fatto che questo sia nullo.

Un secondo problema della *variable expansion* è che se è semplice determinare l'inizio del nome di una variabile, che corrisponde al carattere che segue il "$", non è altrettanto immediato

[13] la pagina di manuale della bash invece fa riferimento a questa funzionalità come alla "Parameter Expansion".

identificarne la fine. Questa nella sintassi esposta viene determinata dal primo carattere non valido per il nome di una variabile. Per cui se ad esempio si definisce la variabile `BASE=prova`, il riferimento a `$BASE_01` non darà come risultato, come qualcuno avrebbe potuto aspettarsi, il valore `prova_01`, ma un valore nullo, perché la variabile `BASE_01` non esiste.

Per risolvere questo tipo di problemi la shell supporta una diversa sintassi per eseguire la *variable expansion*, che prevede l'indicazione esplicita del nome della variabile fra parentesi graffe dopo il "`$`", cioè, per riprendere l'esempio precedente, con qualcosa del tipo:

```
piccardi@anarres:~$ ls -l ${MAIL}
-rw-rw----    1 piccardi mail        4136 Aug 25 17:30 /var/mail/piccardi
```

Questa sintassi, oltre a definire in maniera chiara i caratteri che fanno parte del nome della variabile (nel precedente esempio `${BASE}_01` è ben definito, e da come risultato `prova_01`) consente anche di far eseguire alla shell una serie di operazioni avanzate, come usare un riferimento indiretto (cioè usare una variabile per contenere il nome di quella da referenziare) impostare un valore di default, eseguire espansioni condizionali, ed altre per la cui descrizione dettagliata si rimanda alla sezione "`Parameter Expansion`" della pagina di manuale.

Per poter vedere l'elenco delle variabili di shell che sono state definite (vedremo in sez. 2.1.5 come effettuarne l'impostazione al login o quando si esegue la shell su un terminale) si può usare il comando interno `set`, che ne stampa a video la lista ed i relativi valori nello stesso formato in cui si eseguono le assegnazioni. Il comando `set` però, oltre alle variabili, stampa anche tutte le funzioni (vedi sez. 2.1.5) definite nella shell, per cui la lista ottenuta può essere anche molto lunga e le variabili, che sono all'inizio, si possono perdere di vista.

Si tenga presente che `set` effettua la stampa delle variabili di shell soltanto se invocato senza argomenti (il comando dedicato alla gestione delle variabili in realtà è `declare`, che tratteremo a breve) mentre in generale il comando viene invocato con le opportune opzioni per impostare alcuni comportamenti di default della shell, torneremo su questo in sez. 2.1.6.

Qualora si voglia cancellare una variabile definita in precedenza si può usare invece il comando interno `unset`, seguito dal nome della variabile, il comando è anche in grado, quando usato con l'opzione `-f`, di cancellare una funzione di shell. La modifica di una variabile di shell si effettua semplicemente assegnando alla variabile un nuovo valore. Si tenga presente che benché sia possibile cancellare il contenuto di una variabile assegnandole una stringa vuota, questo non è equivalente a cancellarla con `unset`, dato che essa resterebbe definita, anche se con valore nullo.

Si noti che mentre la shell fornisce la sintassi e gli opportuni comandi interni per assegnare e cancellare la singola variabile, altrettanto non accade per la visualizzazione del contenuto. Ciò avviene perché questo può essere fatto attraverso l'uso di uno dei comandi specialistici della *Unix toolbox*. In particolare il comando da usare è `echo`,[14] il cui solo compito è stampare in uscita la stringa (o le stringhe) che gli vengono passate come argomento: siccome la shell esegue automaticamente l'espansione delle variabili prima di passare gli argomenti ad un comando, si potrà usare `echo` per leggere il contenuto di una variabile con una riga del tipo:

```
piccardi@anarres:~/truedoc/corso$ echo $USER
piccardi
```

[14]si tenga presente che nel caso della `bash` il comando è disponibile anche come comando interno, per cui di default viene eseguita la versione interna fornita dalla shell.

dove la variabile `USER` viene prima *espansa* dalla shell nella stringa `piccardi`, dopo di che quest'ultima viene passata come argomento ad `echo`, che, come avrebbe fatto per una qualunque altra stringa che potreste aver scritto direttamente sulla linea di comando, la stamperà a video.

Il comando `echo` è uno dei più elementari e prende due sole opzioni. La prima è `-n`, che evita la stampa del carattere di a capo alla fine della stringa passata come argomento. Questa opzione è utile quando si vuole costruire una riga di testo con più invocazioni del comando, e la si usa in genere all'interno di uno *script* (tratteremo gli script in sez. 2.1.5), dato che sul terminale si avrebbe l'interferenza da parte del prompt.

La seconda opzione è `-e` che attiva l'interpretazione di espressioni speciali, introdotte dal carattere "\" (come `\n`), che consentono di ottenere la stampa di una serie di caratteri di controllo (nel caso un ritorno a capo). L'elenco completo di queste espressioni, insieme a tutti gli altri dettagli relativi al comando, si trova nella relativa pagina di manuale, accessibile con `man echo`.

La `bash` definisce automaticamente alcune variabili e ne utilizza delle altre qualora siano definite, ed in alcuni casi, se queste non sono state definite, le utilizza impostandole automaticamente ad un valore di default. In questo modo da una parte diventa possibile accedere ad una serie di informazioni che possono essere riutilizzate sulla riga di comando, dall'altra si può modificare il comportamento della shell scrivendo degli opportuni valori nelle opportune variabili.

Un elenco delle principali variabili predefinite riconosciute dalla shell è riportato in tab. 2.4, mentre per l'elenco completo e la descrizione di tutti i loro effetti si può consultare la sezione "`Shell Variables`" della pagina di manuale.

Variabile	**Significato**
`BASH_ENV`	se impostata viene usata nell'esecuzione degli script (vedi sez. 2.1.5) come l'indicazione del file da cui leggere eventuali impostazioni di inizializzazione.
`GLOBIGNORE`	una lista, separata da ":", di nomi di file da ignorare nel *filename globbing*.
`HISTFILE`	il file in cui viene memorizzata la storia dei comandi.
`HISTFILESIZE`	il massimo numero di linee da mantenere nel file della storia dei comandi.
`HISTSIZE`	il numero di comandi da mantenere nella storia dei comandi.
`HOME`	la home directory dell'utente, viene usata anche dal comando interno `cd`.
`HOSTNAME`	il nome della macchina.
`IFS`	elenco dei caratteri che separano in campi diversi il risultato di una *espansione* di shell.
`LANG`	la *localizzazione* utilizzata dai programmi (vedi sez. 3.1.4).
`OSTYPE`	la descrizione del sistema operativo corrente.
`PATH`	la lista delle directory in cui si trovano i comandi.
`PS1`	la stringa che indica il formato del *prompt*.
`PWD`	la directory di lavoro corrente.
`SHLVL`	livello della *subshell* (vedi sez. 2.1.5).
`TERM`	il tipo di terminale su cui si sta operando (usata per cambiare le formattazioni dei dati stampati laddove esistano specifici supporti).
`USER`	l'*username* dell'utente.

Tabella 2.4: Le principali variabili di shell.

Una delle altre funzionalità che la shell fornisce con l'uso delle variabili è quella dell'*ambiente* (in inglese *environment*). Esso è una funzionalità messa a disposizione dal sistema che viene usata per poter utilizzare il valore di alcune variabili di shell anche all'interno dei programmi che questa mette in esecuzione.[15]

Non tutte le variabili definite nella shell sono però inserite nell'*ambiente* in quanto molte di esse (come ad esempio `PS1`) sono di interesse esclusivo della shell. Se si vuole inserire una variabile nell'ambiente, ottenendo quella che si suole chiamare una *variabile di ambiente*, lo si deve fare esplicitamente usando il comando interno `export` seguito dal nome della variabile, che deve essere già definita. La `bash` supporta anche la definizione della variabile nella stessa linea in cui viene esportata nell'ambiente, con una sintassi del tipo `export VAR=val`, questa però è una estensione che non è supportata da altre shell per cui quando si scrivono script è il caso di evitare questa sintassi per mantenere la compatibilità.

Lo stesso risultato si può ottenere con l'uso del comando interno `declare`, che serve in generale ad assegnare variabili e funzioni (tratteremo queste ultime in sez. 2.1.5) o impostarne gli attributi. Il comando può anche essere invocato, con la stessa identica sintassi, come `typeset`. Se usato senza opzioni o argomenti `declare` mostra la lista delle variabili e funzioni di shell definite, esattamente come `set`, in generale però viene usato passando il nome di una variabile come argomento per impostarne le caratteristiche tramite una delle opzioni riportate in tab. 2.5.

Ad esempio si può marcare una variabile come accessibile in sola lettura con l'opzione `-r`, o metterla nell'ambiente con l'opzione `-x`, nel qual caso il risultato è del tutto equivalente all'uso di `export`. In tab. 2.5 si sono riportate solo le opzioni relative alle proprietà delle variabili, il comando opera anche sulle funzioni di shell (trattate in sez. 2.1.5) e per la documentazione completa si può fare riferimento alla descrizione nella sezione "`SHELL BUILTIN COMMANDS`" della pagina di manuale. Per la rimozione di una variabile dall'ambiente, senza perderne il valore, si può usare invece la sintassi `declare +x`. Si tenga presente infine che se si cancella una variabile con `unset` questa viene automaticamente rimossa anche dall'ambiente.

Opzione	**Significato**
`-a`	dichiara una variabile vettoriale.
`-i`	dichiara una variabile intera.
`-r`	rende la variabile accessibile in sola lettura (e ne impedisce la cancellazione con `unset`).
`-x`	inserisce la variabile nell'ambiente.
`-p`	stampa tutti gli attributi associati ad una variabile o una funzione.

Tabella 2.5: Principali opzioni del comando interno `declare`.

Per visualizzare le variabili di shell presenti nell'ambiente si può usare lo stesso `export` senza specificare nulla, e si otterrà la lista delle variabili presenti nell'ambiente, precedute dalla dichiarazione "`declare -x`"; la lista infatti viene stampata in un formato pronto per essere salvato su un file ed usato come script (tratteremo gli script in sez. 2.1.5) per ricreare l'ambiente.

[15] in sostanza quando un programma viene messo in esecuzione dalla *system call* `exec` deve ricevere una serie di informazioni; una parte di queste sono gli argomenti, che vengono presi direttamente da quanto scritto nella riga di comando con il meccanismo illustrato in precedenza, l'altra parte sono appunto le *variabili di ambiente*, che fanno parte dell'*environment*, per una trattazione dettagliata dell'argomento si veda sez. 2.3.3 di [GaPiL].

Se si vuole solo una semplice lista di nomi con il relativo valore si può usare il comando `env`. Questo comando in generale permette di eseguire un altro comando, da passare come argomento, fornendogli un ambiente modificato rispetto a quello corrente della shell; con l'opzione `-i` infatti l'ambiente viene completamente svuotato, mentre con una serie di opzioni `-u` si possono cancellare altrettante variabili di ambiente (da specificare come parametro per l'opzione). Se si invoca `env` senza argomenti il comando si limita a stampare la lista delle variabili di ambiente, nella classica forma `VARIABILE=valore`.

Le variabili di ambiente sono di grande importanza perché sono usate in moltissimi casi per controllare alcuni comportamenti predefiniti dei programmi. Per questo l'uso di `env` è in genere una buona misura di sicurezza per fornire ai comandi che si eseguono un ambiente noto e ben definito cancellando preventivamente quanto eventualmente presente. Molti comandi infatti sono controllati anche dai valori delle variabili di ambiente, ed eseguendoli in un ambiente che può essere stato "*manomesso*" si potrebbero avere risultati spiacevoli.

Alcune impostazioni e valori di uso generale vengono allora memorizzati in una serie di variabili che di norma ogni shell deve definire, come `USER`, che indica il nome dell'utente corrente, `HOME` che ne indica la *home directory*, `TERM` che specifica il tipo di terminale su cui si sta operando, `LANG` che indica la localizzazione (vedi sez. 3.1.4) in uso, `PATH` che specifica la lista delle directory dove cercare i comandi; altre variabili sono invece opzionali, ma sono riconosciute dalla gran parte dei programmi, come `EDITOR` che indica quale programma usare come editor predefinito o `BROWSER` che indica quale navigatore per il web utilizzare.

Una variabile molto importante ma poco nota, ed il cui significato spesso viene frainteso, è `IFS`, il cui nome sta per *Internal Field Separator*. Questa variabile viene utilizzata per indicare la lista di caratteri che devono essere considerati come separatori nei risultati di una *espansione* di shell ed il suo valore di default è impostato alla sequenza spazio, tabulatore, ritorno a capo. La variabile è importante perché ha un effetto diretto, la cui comprensione non è proprio immediata, sul risultato dell'interpretazione della riga di comando.

Come abbiamo già visto con la *variable expansion* la shell consente di sostituire automaticamente sulla riga di comando alcune espressioni con il risultato della relativa *espansione* (nel caso citato una variabile con il suo contenuto). Esistono anche, e le tratteremo a breve, altre espansioni, come una lista di file ottenuta con l'uso dei caratteri jolly del *filename globbing*, o il risultato di un calcolo dato da una *arithmetic expansion* o l'output di un altro comando fornito da una *command expansion*.

In tutti questi casi il risultato della espansione deve essere di nuovo sottoposto ad una scansione per suddividerlo in una lista di campi che verranno interpretati come argomenti o opzioni; questo viene fatto usando come separatori i caratteri che sono elencati in `IFS`, in questo modo se si ha un comando che genera delle liste usando un carattere di separazione che non rientra nei classici caratteri di spaziatura, si può aggiungerlo a `IFS` per usarlo come separatore.

Si tenga presente però che `IFS` non ha nessun effetto sulla scansione di una linea di comando ordinaria che non contiene espansioni. Infatti nella costruzione degli argomenti prima viene effettuata una scansione iniziale, per la quale vengono usati come separatori solo i caratteri di spaziatura, una eventuale *espansione di shell* viene sempre effettuata in un secondo passo e solo a questo punto possono entrare in gioco i caratteri elencati in `IFS`.

Un'altra variabile di particolare rilevanza è `LANG`; su Linux infatti è presente il supporto, attraverso l'uso di una infrastruttura generica che fa parte delle librerie di sistema, per la localizzazione dei programmi sia in termini di lingua e set di caratteri utilizzati nella comunicazione

con gli utenti, che per altri aspetti che riguardano convenzioni nazionali, come il formato dell'ora, della valuta, ecc. Come spiegato in maggior dettaglio in sez. 3.1.4, questi possono essere scelti in maniera generale sulla base del valore di questa variabile, o per i singoli aspetti secondo i valori delle singole variabili a questo deputate (che, come vedremo in sez. 3.1.4, sono tutte nella forma `LC_*`).

Al di là della stampa dei vari messaggi in italiano (o in ogni eventuale altra lingua selezionata) anziché in inglese, uno degli aspetti più significativi dell'uso di `LANG`, ed il motivo per cui ne accenniamo anche in questa sede, è che questa nelle versioni più recenti dei programmi influenza anche le modalità con cui essi eseguono l'ordinamento alfabetico. Infatti nella localizzazione di default (quella per cui `LANG=C`) l'ordine delle lettere prevede prima tutte le maiuscole e poi tutte le minuscole, cioè qualcosa del tipo `ABC...Zabc...z`, mentre ad esempio in italiano l'ordinamento è completamente diverso e prevede la minuscola seguita della corrispondente maiuscola, cioè qualcosa del tipo `aAbBcC...zZ`.

Questo significa che a seconda del supporto o meno per questa funzionalità e del valore di `LANG` si possono ottenere risultati completamente diversi quando i programmi o la shell stessa hanno a che fare con ordinamenti alfabetici. Pertanto può risultare opportuno, quando si vuole essere sicuri di avere un comportamento uniforme, impostare forzatamente questa variabile al valore "`C`" (torneremo sul significato dei valori possibili in sez. 3.1.4) di modo che venga usato sempre il comportamento di default previsto anche dallo standard POSIX, che è quello che in seguito considereremo come default.

Di altrettanta se non maggiore rilevanza è la variabile `PATH` che è direttamente connessa ad una delle funzionalità della shell su cui finora abbiamo sorvolato, dandola per scontata: quella del cosiddetto *path search*, cioè del fatto che la shell permette di associare automaticamente un nome, scritto sulla linea di comando, ad un programma da lanciare. In genere, a meno dell'uso degli *alias* su cui torneremo a breve, questo nome deve corrispondere al nome del file che contiene il programma che la shell deve eseguire. In sostanza, come visto nell'esempio iniziale di sez. 2.1.3, si può sempre scrivere solo il nome del file al posto del pathname completo, se cioè si vuole leggere una pagina di manuale, basterà scrivere `man` e non sarà necessario specificare `/usr/bin/man`.

Si può fare così perché la shell esegue automaticamente una ricerca nel cosiddetto *PATH dei comandi*; cioè dato un nome sulla linea di comando la shell cerca un file da eseguire con quel nome nella lista di directory contenute nella variabile d'ambiente `PATH`. Se si vuole evitare questo comportamento, ad esempio perché il programma che si vuole lanciare non è in una di quelle directory e non sarebbe trovato, è necessario specificare, sia in forma assoluta che relativa, il pathname per accedere al comando, la shell riconosce che si vuole fare questo quando nel nome del comando compare il carattere "`/`", che indica una directory. Ad esempio per lanciare un comando (o uno script) nella directory corrente si usa la sintassi:

```
./comando
```

La variabile `PATH` ha la forma di una lista di pathname di directory, separati fra loro da un carattere di due punti, un formato comune usato nelle variabili di ambiente tutte le volte che si deve specificare una lista di directory in cui effettuare ricerche (lo reincontreremo spesso, ad esempio in sez. 3.1.2 per le directory dove sono mantenute le librerie di sistema). Un esempio del valore di questa variabile potrebbe essere il seguente:

```
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
```

come è presa dalla definizione standard per un utente comune su Debian. Si noti come in questo caso siano indicate solo le directory che, nella standardizzazione del filesystem illustrata in sez. 1.2.3, contengono i programmi di uso per l'utente; sono pertanto assenti directory come `/sbin`, `/usr/sbin`, ecc. che usualmente vengono inserite dentro `PATH` soltanto per l'amministratore.

Questo è il motivo per cui alcuni comandi "*non funzionano*" quando chiamati da utente normale, in realtà non è che non funzionano, solo non vengono trovati i relativi programmi, che potrebbero tuttavia essere eseguiti ugualmente (con privilegi ridotti ovviamente) usando un pathname assoluto o inserendo le directory in cui si trovano all'interno di `PATH`.

Si tenga conto che `PATH` è comunque modificabile dall'utente, che può personalizzarla a piacere, ed aggiungere un'altra directory, come ad esempio la directory corrente, che di norma non vi è mai inclusa. Questa scelta ha una ragione precisa: se tenete dentro `PATH` la directory corrente un altro utente potrebbe mettere nelle directory cui ha accesso una versione "*opportunamente*" modificata di un comando di sistema, che verrebbe eseguita al posto dell'originale, col rischio di guai seri qualora ciò avvenisse per l'amministratore; è vero che se si inserisce la directory corrente in coda a `PATH` i comandi standard hanno la precedenza, ma qualcuno potrebbe sempre creare un programma `la` e aspettarvi al varco per un comune errore di battitura di `ls`.

Qualora si voglia cambiare `PATH` il dubbio casomai può essere sul come farlo senza doversi riscrivere tutto da capo il contenuto precedente; per questo ci viene di nuovo in aiuto la sintassi della riga di comando, per cui è possibile ridefinire una qualunque variabile con una espressione del tipo:

```
PATH=$PATH:./
```

dove di nuovo si è usato la capacità della shell di *espandere* il valore delle variabili, che funziona anche all'interno della sintassi usata per assegnare le stesse. Si noti come una riga del genere non esegua nessun programma, limitandosi ad utilizzare esclusivamente le funzionalità interne della shell. Una delle caratteristiche peculiari di `bash` è quella di essere in grado di vedere immediatamente i nuovi programmi non appena si ridefinisce il valore di `PATH`, mentre con altre shell (ad esempio la `csh`) può essere necessario utilizzare un apposito comando interno (nel caso `rehash`).

Come complemento alla gestione del *path search* la shell fornisce il comando interno `which` che, dato il nome di un programma presente nel *path search*, ne stampa a video il pathname assoluto, ad esempio potremo ottenere il pathname del comando `ls` con:

```
piccardi@hain:~/Truelite$ which ls
/bin/ls
```

in questo modo è possibile anche capire quale sia effettivamente il programma che viene usato qualora ne esistano più versioni in alcune delle directory contenute all'interno di `PATH`. La regola è che viene usato sempre il primo programma che viene trovato nella scansione delle directory del *path search*; questa avviene da sinistra a destra nell'ordine in cui le directory sono elencate all'interno di `PATH`. Se non viene trovato nessun programma corrispondente, non sarà stampato nulla.

Si tenga presente però che `which` funziona controllando se il nome passato come argomento compare nella lista dei programmi contenuti nelle directory elencate in `PATH`, se si indica il nome di un comando interno della shell questo non viene considerato. Per questo, se si vuole capire se

si sta usando un comando interno o un programma o un *alias* (che tratteremo a breve) si deve usare un altro comando interno, `type`, nel qual caso otterremo qualcosa del tipo:

```
piccardi@hain:~/Truelite$ type export
export is a shell builtin
```

Un'ulteriore funzionalità che la shell mette a disposizione è quella degli *alias*: prima ancora di cercare se il nome di un comando corrisponde ad un programma presente in una delle directory indicate in `PATH`, la shell controlla se esso corrisponde ad un *alias*; ci permette così di ideare nuovi comandi, definire abbreviazioni per quelli più usati, o ridefinire lo stesso nome di un comando per farlo comportare in maniera diversa.

Un *alias* si crea con il comando interno `alias` associando un nome ad un altro comando (che può a sua volta essere stato definito in un altro *alias*) con una sintassi del tipo:

```
alias ll='ls --color=auto  -l'
```

ma si può anche definire una abbreviazione con:

```
alias l='ls -l'
```

o ridefinire un comando esistente con:

```
alias rm='rm -i'
```

In questo modo nel primo caso si definisce una versione colorata e "*prolissa*" di `ls`, nel secondo una abbreviazione per l'uso di `ls -l` e nel terzo si ridefinisce `rm` in modo che di default sia richiesta conferma nella cancellazione dei file. Per cancellare un *alias* si può usare il comando `unalias` seguito dal nome dello stesso. Qualora invece si esegua il comando `alias` senza specificare nulla questo mostrerà la lista degli *alias* già definiti.

Un'altra funzionalità di grande utilità fornita della shell è il cosiddetto *filename globbing*. Si può cioè operare su insiemi di file con nomi simili passando ai comandi che dovranno operare su di essi degli argomenti che li indicano in gruppo, tramite i cosiddetti *caratteri jolly* (in inglese *wildcard*). Se cioè all'interno di una riga di comando si usa come argomento una espressione che contiene uno di questi caratteri jolly, la shell lo interpreterà come una richiesta di *filename globbing* ed interpreterà quella espressione come indicante un insieme di file.

Il meccanismo prevede che la shell verifichi quali file corrispondono all'espressione e la *espanda* in una lista che verrà passata al comando al posto del singolo argomento. Normalmente la ricerca viene fatta nella directory corrente ma si può usare il carattere "`/`" per indicare un *pathname* e fare riferimento ad altre directory.

Come nel caso del DOS o del VMS, il carattere "`*`" indica un numero arbitrario di caratteri qualsiasi mentre il carattere "`?`" indica un solo carattere qualunque. Ma oltre a queste due *wildcard* elementari la shell ne supporta alcune più sofisticate. Ad esempio si possono indicare elenchi di possibili alternative per un singolo carattere, ponendole fra due parentesi quadre. Le alternative possono anche essere espresse con una lista dei caratteri voluti messi uno di seguito all'altro, mentre si possono specificare degli intervalli (corrispondenti a tutti i caratteri compresi fra i due estremi) usando due caratteri separati da un "`-`".

In questo caso però occorre fare attenzione al tipo di ordinamento alfabetico dovuto alle diverse localizzazioni, perché a seconda di questo si potranno ottenere risultati diversi a causa

della diversa interpretazione dell'intervallo, ad esempio in italiano l'intervallo `[a-z]` corrisponde a tutte le lettere maiuscole e minuscole tranne la "`Z`" mentre in inglese corrisponderebbe a tutte le minuscole. Nel seguito si farà riferimento al comportamento di default dettato dallo standard POSIX, relativo al caso in cui `LANG=C` (per ulteriori dettagli si rimanda a sez. 3.1.4).

Si può inoltre invertire la selezione dei caratteri specificati fra parentesi quadre precedendo la lista o l'intervallo con un carattere di "`^`". Infine come ultima modalità di espansione si può utilizzare il carattere "`~`" per indicare la home directory dell'utente corrente, mentre si può usare la notazione `~username` per indicare la home directory di un altro utente. Infine qualora l'espressione non corrisponda a nessun file essa verrà passata letteralmente al comando, con in genere il risultato di ottenere un errore di file non esistente.

Per capire meglio facciamo alcuni esempi di *filename globbing*: supposto che per essi venga usata la solita estensione `.tex`, si potranno vedere tutti i sorgenti Latex presenti in una directory con una riga di comando come:

```
piccardi@anarres:~/truedoc/sicurezza$ ls *.tex
cryptoapp.tex  firewall.tex  ids.tex  introsec.tex  sicurezza.tex
```

mentre si potranno selezionare tutti i file il cui nome è lungo esattamente sette caratteri con una riga di comando del tipo:

```
piccardi@anarres:~/truedoc/sicurezza$ ls ???????
ids.aux  ids.tex
```

se invece vogliamo vedere i file i cui nomi finiscono in `c` o in `g` potremo usare:

```
piccardi@anarres:~/truedoc/sicurezza$ ls *[cg]
firewall.log  sicurezza.blg  sicurezza.log
firewall.toc  sicurezza.ilg  sicurezza.toc
```

mentre per selezionare i file il cui nome inizia con una lettera maiuscola potremo usare:

```
piccardi@anarres:~/truedoc/sicurezza$ ls [A-Z]*
Makefile  SicurezzaGL.txt  TODO
```

infine se ci interessano i file LaTeX il cui nome non inizia con una lettera minuscola fra "`d`" e "`r`" potremo usare:

```
piccardi@anarres:~/truedoc/sicurezza$ ls [^d-r]*.tex
cryptoapp.tex  sicurezza.tex
```

Si tenga presente che il meccanismo del *filename globbing* opera esattamente nella modalità descritta, sostituendo all'espressione sulla riga di comando il suo risultato, per cui al comando verranno passati come argomenti i nomi dei file che vi corrispondono, o il nome, se è uno solo, o la stringa letterale se niente corrisponde. Questo comporta che il risultato poi dipenderà da come il comando interpreta gli argomenti che riceve, cosa che alle volte può causare risultati non troppo intuitivi. Ad esempio si fosse usata la seguente riga di comando:

```
piccardi@anarres:~/truedoc/corso$ ls ??????
IPSEC_modes.dia    esp_scheme.pdf  iptraf5.png        openvpn.dia
IPSEC_modes.eps    etherape1.eps   iptraf6.eps        openvpn.eps
IPSEC_modes.pdf    etherape1.png   iptraf6.png        openvpn.pdf
...
```

si sarebbe ottenuto un risultato forse inaspettato, dato che non riporta file di sei lettere. Il risultato è questo perché l'unico nome di file di sei lettere presente nel caso dell'esempio è quello della directory `images`, per cui dall'espansione del *filename globbing* otteniamo una riga di comando equivalente a "`ls images`"; quindi ci verrà stampato, secondo la sintassi di `ls`, il contenuto di tale directory.

Il *filename globbing* e l'uso delle variabili di ambiente sono funzionalità molto utili, però talvolta presentano degli inconvenienti. Ad esempio se si volesse passare ad un comando come argomento un file il cui nome contiene uno dei caratteri jolly, o il carattere "`$`" usato per referenziare le variabili, si avrebbe il problema che la shell cercherebbe di espandere la relativa espressione. Un inconveniente analogo ci sarebbe per i nomi dei file che contengono degli spazi: dato che lo spazio viene usato per separare fra di loro gli argomenti, un tale nome verrebbe spezzato in due con il risultato di passare al comando due argomenti invece di uno.

Per risolvere questi problemi esistono dei caratteri che permettono di disabilitare del tutto o in parte le capacità della shell di interpretare in maniera speciale altri caratteri. Uno di questi è la barra trasversa "\" che anteposta ad un carattere qualsiasi ne disabilita l'interpretazione, così si può usare un asterisco in un argomento indicandolo con "*" o si può utilizzare uno spazio proteggendolo con "\␣" o specificare la stessa barra trasversa con "\\" ed anche, come accennato in precedenza, proteggere il carattere di ritorno a capo, che di norma viene usato come terminazione della riga di comando.

Una modalità alternativa è quella che permette di specificare degli argomenti che vanno considerati come una stringa letterale (risolvendo anche così il problema degli spazi) e per i quali non si vuole che la shell effettui delle espansioni. Questo può essere fatto in due modi, il primo, che disabilita il *filename globbing* e l'interpretazione degli spazi, è quello di definire la stringa racchiudendola fra virgolette (usando cioè il carattere "`"`"), così che al suo interno si possano usare spazi, asterischi e punti interrogativi. In questo modo però resta attiva l'espansione di eventuali variabili di shell tramite il carattere "`$`". Si può eliminare anche questa se al posto delle virgolette si usano gli apici singoli (cioè il carattere "`'`"); in tal caso la stringa sarà interpretata in maniera assolutamente letterale.

Un'altra funzionalità molto utile della shell è quella che viene chiamata *command expansion*; si può cioè usare come argomento sulla riga di comando il risultato ottenuto con un'altra riga di comando. Per fare questo si hanno a disposizione due sintassi equivalenti; la prima prevede che si racchiuda il comando da eseguire fra due apici inversi (usando cioè il carattere "`` ` ``"), la seconda che si usi il costrutto `$(comando)`; ad esempio se nel file `elenco` si è scritto una lista di file si potrà effettuarne la cancellazione con una linea di comando come:

```
piccardi@anarres:~/truedoc/corso$ rm `cat elenco`
rm: cannot lstat 'pippo.tex': No such file or directory
rm: cannot lstat 'vecchio.tex': No such file or directory
```

che nel caso fallisce perché i file elencati non esistono.

In questo caso quello che succede è che la shell esegue il comando racchiuso fra apici inversi, e mette il suo output sulla riga di comando, come se lo si fosse scritto manualmente. Questa in realtà è una semplificazione, per essere precisi quello che avviene è che l'uscita del programma viene scansionata usando i caratteri di `IFS` come separatori e poi "*espansa*" in un elenco di argomenti per il programma. La *command expansion* ci mette a disposizione un mezzo estremamente potente per collegare fra loro i comandi: possiamo usare il risultato di uno di essi per generare gli argomenti di un altro.

Simile alla *command expansion* è la cosiddetta *arithmetic expansion* che permette di effettuare dei semplici calcoli numerici. Per farlo si usa una sintassi del tipo di `$((espressione))`, dove con `espressione` si indica una espressione aritmetica da calcolare. L'espressione usa numeri interi e gli stessi operatori aritmetici usati dal linguaggio C come `+`, `-`, `*` e `/`, la descrizione completa di tutti gli operatori e della sintassi delle espressioni aritmetiche è disponibile nella pagina di manuale della `bash`, alla sezione "`ARITHMETIC EVALUATION`". Il risultato del calcolo sarà riportato direttamente come argomento, e potremo avere un qualcosa del tipo:

```
piccardi@hain:~/truedoc/corso$ echo $((5+7))
12
```

e considerato che all'interno dell'espressione si può usare sia la *variable expansion* che la *command expansion*, si capisce anche come da questa espansione si possano trarre ulteriori potenzialità di utilizzo.

L'ultima funzionalità della shell che prenderemo in esame è quella della *storia dei comandi*, la cosiddetta *history*, che permette di accedere, usando i tasti di freccia in alto e in basso, alle linee di comando eseguite in precedenza. La `bash` infatti mantiene in memoria le linee di comando che ha eseguito e poi quando esce salva l'elenco in un apposito file.

Di default la storia dei comandi viene salvata nel file `.bash_history` nella home dell'utente, ma questo può essere cambiato in qualunque altro file specificato dalla variabile `HISTFILE`. La shell di default mantiene in memoria un massimo di 500 comandi, ma questo valore può essere modificato specificando un nuovo default nella variabile `HISTSIZE`, si può anche indicare una dimensione massima in righe del file `.bash_history` con la variabile `HISTFILESIZE`.

Diventa così possibile non solo rieseguire i comandi precedenti, navigando avanti ed indietro lungo la storia con i tasti di freccia in alto e in basso, ma è anche possibile effettuare una ricerca incrementale nella *history* utilizzando la combinazione `C-r` seguita dal testo della ricerca (il cui risultato comparirà all'interno di un apposito *prompt*). Inoltre con l'uso del carattere "`!`" diventa possibile richiamare una specifica voce sulla riga di comando stessa, indicandola sia per numero che tramite le prime lettere, secondo le modalità illustrate in tab. 2.6, utilizzando quella che viene chiamata *history expansion*.

Sintassi	**Significato**
`!!`	richiama il comando più recente.
`!n`	richiama il comando numerato `n` nella *history*.
`!-n`	richiama il comando precedente di `n` posizioni nella *history*.
`!string`	richiama l'ultimo comando eseguito che inizia per `string`.
`!?string`	richiama l'ultimo comando eseguito che contiene la stringa `string`.
`^string^subst`	richiama il comando precedente sostituendo `string` con `subst`.

Tabella 2.6: Le modalità di utilizzo della *history expansion*.

Con il comando `history` inoltre si può visualizzare l'intero contenuto della *history*, in cui ogni riga è numerata progressivamente, con il valore che si può usare con "`!`" per richiamarla. Così ad esempio per richiamare righe di comando precedenti si potrà fare qualcosa come:

```
piccardi@anarres:~/truedoc/corso$ which chown
/bin/chown
piccardi@anarres:~/truedoc/corso$ ls -l $(!wh)
ls -l $(which chown)
-rwxr-xr-x    1 root     root        19948 Aug 19 03:44 /bin/chown
```

mentre se si vuole usare direttamente il numero, si potrà verificarlo con:

```
piccardi@anarres:~/truedoc/corso$ history
...
  523  ls [^d-r]*.tex
  524  which chown
  525  ls -l $(which chown)
  526  history
```

e richiamare un comando con:

```
piccardi@anarres:~/truedoc/corso$ !526
...
  523  ls [^d-r]*.tex
  524  which chown
  525  ls -l $(which chown)
  526  history
  527  history
```

Si tenga infine conto che il carattere "`!`", come mostrato nel primo esempio, viene espanso quando usato nella linea di comando, e per usarlo in maniera letterale occorre o proteggerlo con la barra rovesciata, o metterlo in una stringa delimitata da apici singoli. Per maggiori dettagli riguardo tutto l'argomento si può fare riferimento alla sezione "`HISTORY EXPANSION`" del manuale della shell.

Concludiamo questa sezione relativa alla funzionalità di base della shell con una panoramica più estesa sull'uso interattivo della riga di comando, ed in particolare sulle cosiddette "*scorciatoie da tastiera*" disponibili. Dato che la gran parte di queste attengono proprio all'uso della *history* ci è parso opportuno trattarle dettagliatamente a questo punto.

Combinazione	**Significato**
`C-r`	ricerca incrementale nella history.
`C-a`	spostamento a inizio riga.
`C-e`	spostamento a fine riga.
`C-u`	cancellazione da posizione corrente a inizio riga.
`C-k`	cancellazione da posizione corrente a fine riga.
`C-y`	inserisce l'ultimo testo cancellato.
`M-b`	spostamento indietro di una parola.
`M-f`	spostamento avanti di una parola.
`C-l`	ripulitura e posizionamento in testa allo schermo.
`Tab`	esegue l'*autocompletamento* del testo corrente.

Tabella 2.7: Le scorciatoie da tastiera per l'*editing* di linea.

Abbiamo visto in sez. 2.1.2 come alcune combinazioni di tasti (vedi tab. 2.1) siano collegate, grazie all'interfaccia dei terminali, all'emissione di segnali o altre funzionalità che consentono di controllare il comportamento dell'interfaccia a riga di comando. La shell stessa poi supporta, come per il `C-r` di cui abbiamo appena parlato, l'uso di altre combinazioni di tasti che consentono di effettuare modifiche sulla riga di comando che si sta scrivendo; ad esempio con `C-a` si può andare a inizio riga e con `C-e` a fine riga.

Queste funzionalità derivano in realtà da una libreria di gestione chiamata appunto *readline* che consente di operare in maniera generica su una riga di terminale, spostandosi modificando e cancellando parti utilizzando una serie di combinazioni di tasti che possono essere configurate

nei dettagli (torneremo su questo in sez. 2.1.6) ma che di default usano le stesse convenzioni dell'editor `emacs` che illustreremo brevemente in sez. 2.3.2.

Si sono riportate le principali combinazioni di tasti forniti da questa libreria in tab. 2.7. Nella tabella con "`M`" si intende il tasto *alt*, la notazione con cui vengono espresse le combinazioni di tasti verrà trattata più dettagliatamente in sez. 2.3.2. Per l'elenco completo di tutte le funzionalità si può consultare la sezione "`Readline Command Names`"della pagina di manuale della shell.

Una di queste funzionalità però, quella dell'*autocompletamento* con l'uso del tasto di tabulazione, merita di essere illustrata a parte, in quanto costituisce una delle caratteristiche più avanzate della `bash`. La shell infatti supporta nativamente il completamento dei comandi e dei nomi dei file che si scrivono sulla riga di comando. Basta cioè inserire le prime lettere di un comando e premere il tabulatore per avere il completamento del nome del comando, o, se c'è ambiguità con più completamenti possibili, il completamento della parte non ambigua; in quest'ultimo caso inoltre, premendo il tabulatore due volte di fila, si potrà avere la lista dei completamenti possibili.

Questo meccanismo funziona di default con i programmi per la prima parola scritta sulla riga di comando, e con i nomi dei file per le parole successive, ma la vera potenza della shell è che l'autocompletamento è completamente programmabile (si consulti la sezione "`Programmable Completion`" della pagina di manuale) e può essere esteso nelle modalità più varie. La gran parte delle distribuzioni definisce una serie di queste estensioni nel file `/etc/bash_completion`, usando il quale diventa possibile ottenere completamenti, anche molto complessi, sui possibili argomenti e opzioni di vari comandi.

2.1.4 La redirezione dell'I/O

Si è dedicata una sezione separata a questa funzionalità della shell per la sua particolare rilevanza, dato che è proprio la redirezione delle operazioni di I/O che ci permette di costruire quella catena di montaggio di cui abbiamo parlato in sez. 2.1.1, che consente di combinare le capacità dei vari comandi ed ottenere l'effettiva potenza della riga di comando.

Come accennato in sez. 2.1.2 in un sistema unix-like i programmi si aspettano di avere disponibili e già aperti tre file: *standard input*, lo *standard output* e lo *standard error*. Seguendo una convenzione comune, i comandi si aspettano di ricevere i loro dati in ingresso sullo *standard input* (e non da uno specifico file o dispositivo, a meno che questo non sia previsto come argomento), mentre scrivono i loro dati in uscita sullo *standard output*. Lo *standard error* in caso di operazioni regolari non viene mai usato; su di esso vengono scritti solo gli eventuali messaggi di errore.

Quando si usa l'interfaccia a riga di comando *standard input*, *standard output* e *standard error* vengono aperti nella procedura di accesso al sistema e sono tutti associati al terminale su cui viene eseguita la shell. Dato che, come accennato in sez. 1.3.1, i file aperti vengono ereditati dai processi figli, normalmente quando la shell lancia un comando questi file restano gli stessi anche per lui, che pertanto potrà ricevere i dati in ingresso dalla tastiera e scrivere i suoi risultati sullo schermo.

Gli eventuali errori vanno anch'essi sul terminale e compaiono sullo schermo insieme ai dati in uscita, ma questo avviene solo perché si è usato lo stesso file dello *standard output*. In generale, ad esempio quando si redirige lo *standard output* come vedremo fra poco, non è così essendo i due file distinti, per cui si potrà mandare l'uscita di un comando su un file mentre gli errori

resteranno sul terminale. La *redirezione dell'I/O* infatti è la capacità della shell di modificare, per il processo figlio con cui esegue un comando, i file a cui fanno riferimento lo *standard input*, lo *standard output* e lo *standard error*, così che esso possa ricevere dati in ingresso o scrivere dati in uscita ed errori su degli altri file e non sul terminale.

Per capire meglio il significato della redirezione prendiamo come esempio il comando `cat`. Questo è un comando elementare che serve a leggere uno o più file in ingresso, passati come argomenti, e a scriverne il contenuto sullo *standard output*. Se non si specifica nessun file come argomento il comando legge di default dallo *standard input*, per cui se eseguiamo:

```
piccardi@anarres:~$ cat
```

ci troveremo in una situazione in cui il comando si blocca sulla riga successiva in attesa che scriviamo qualcosa sul terminale. Se lo facciamo scrivendo `prova` seguito da invio otterremo qualcosa del tipo:

```
piccardi@anarres:~$ cat
prova
prova
```

dove il comando reagisce all'invio ristampando sul terminale quanto appena scritto e attendendo nuovi dati in ingresso.

Questo avviene in quanto sul terminale si ha quello che si chiama *I/O bufferizzato* ed i dati in ingresso vengono letti e scritti una riga alla volta, per cui alla pressione dell'invio verrà letto quanto appena scritto, ed il comando lo riscriverà sullo *standard output*; per uscire dal comando occorre nel caso indicare che la lettura è conclusa, ad esempio inviando un end-of-file sull'input con `C-d`, o interrompere il programma con un segnale.

Usato così il comando non è di grande utilità, dato che in genere non serve a molto scrivere qualcosa sulla tastiera per vederselo ristampare sul terminale ad ogni riga; però se vogliamo vedere il contenuto di un file il modo più immediato per farlo è con un comando del tipo:

```
piccardi@anarres:~$ cat /etc/profile
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

PATH="/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games"
...
```

dato che il comando leggerà il contenuto del file e lo riverserà sullo *standard output*, ed essendo quest'ultimo collegato al terminale, lo si vedrà sullo schermo.

Fin qui l'utilità del comando è relativa, dato che programmi per visualizzare il contenuto del file ce ne sono altri, magari più sofisticati come `more` e `less`, che permettono anche di vedere il file un pezzo alla volta, e andare avanti e indietro, senza scrivere tutto il file in un colpo solo sul terminale, per cui l'inizio va perso nello scorrimento dello schermo. Il comando comunque prende alcune opzioni per facilitare la visualizzazione, come `-n` che stampa un numero progressivo per ogni riga, `-t` che stampa i tabulatori come `^I`, `-v` che visualizza i caratteri non stampabili, e `-E` che scrive un `$` alla fine di ogni riga. La descrizione completa si trova al solito nella pagina di manuale, accessibile con `man cat`.

Ma la vera utilità di `cat`, il cui scopo in realtà non è quello di mostrare il contenuto di un file, emerge solo quando esso viene unito alla capacità della shell di modificare i file associati a

standard input, *standard output* e *standard error*, cioè con la *redirezione dell'I/O*. Per fare questo la shell riconosce sulla riga di comando alcuni caratteri che svolgono la funzione di operatori, detti appunto *operatori di redirezione*. I due operatori più elementari sono "`<`" e "`>`" che permettono rispettivamente di redirigere lo *standard input* e lo *standard output* sul file specificato dall'utente dopo l'operatore.

In realtà si può effettuare la redirezione di un qualunque file descriptor usato da un programma, specificando il suo numero prima dell'operatore, questo comporta ad esempio che per redirigere lo *standard error* si usa l'operatore "`2>`" dato che esso è associato al file descriptor 2, ma si potrebbe utilizzare allo stesso modo "`3>`" per redirigere il file descriptor 3; i dettagli sulle varie forme di redirezione si possono trovare nella pagina di manuale alla sezione "`REDIRECTION`".

Cominciamo allora a vedere come si usano questi due operatori. Una modalità alternativa di stampare il contenuto di un file con `cat` è utilizzare un comando del tipo:

```
piccardi@anarres:~$ cat < /etc/profile
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).
...
```

in cui invece di leggere un file specificato come argomento sulla riga di comando (si noti che qui `cat` non ha argomenti, il nome del file è dopo il carattere "`<`"), si torna a leggere dallo *standard input* che però in questo caso, invece di essere associato al terminale, è stato rediretto dall'operatore "`<`" su `/etc/profile`, per cui si otterrà il contenuto di questo file.

La redirezione dell'output avviene in maniera identica associando con "`>`" lo *standard output* ad un file. Ad esempio si potrà copiare il contenuto del file `pippo` sul file `pippo2` utilizzando un comando del tipo:

```
cat pippo > pippo2
```

dato che con questa redirezione invece che sul terminale il contenuto di `pippo` sarà scritto su `pippo2`.

Fin qui niente di particolarmente utile, dato che nel primo caso si sarebbe potuto leggere direttamente il file, mentre nel secondo si sarebbe potuto usare direttamente `cp`. Però, e qui emerge anche la vera utilità del comando `cat`, con `cp` si può solo copiare il contenuto di un file su un altro, ma cosa succede invece se con `cat` specifichiamo come argomenti una serie di file da leggere e poi redirigiamo l'uscita su di un altro file?

Per quanto abbiamo detto in questo caso `cat` leggerà il contenuto di ciascuno dei file passati come argomenti, riscrivendo il tutto sullo *standard output*. Avendo rediretto lo *standard output* su un altro file il risultato sarà che quest'ultimo avrà come contenuto la *concatenazione* del contenuto di tutti i file passati come argomenti a `cat`, ed infatti il nome del comando, non proprio intuitivo, deriva da *con**cat**enate file*.

Così se si è spezzato un file di grosse dimensioni in una serie di file più piccoli (ad esempio per scriverlo su una serie di CD) si potrà ricostruirlo con un semplice comando del tipo:

```
cat pezzo1 pezzo2 pezzo3 ... > filecompleto
```

La redirezione dello *standard input* è invece molto utile con i programmi che richiedono l'immissione dei dati da tastiera, specie quando questi vengono eseguiti in uno script. Una volta definiti quali dati sono necessari infatti li si potranno scrivere in un file, e farli leggere al programma semplicemente con la redirezione dello *standard input*.

Si tenga conto che quando si effettua la redirezione dello *standard output* se il file su cui la si esegue non esiste esso viene creato, mentre se esiste il suo contenuto viene sovrascritto, a meno che non si sia bloccato questo comportamento indicando alla shell di segnalare un errore con `set -C` (vedi tab. 2.12), occorre pertanto stare molto attenti a non cancellare dei dati. Per questo è bene essere consapevoli della modalità con cui il meccanismo funziona, se infatti si pensasse di aggiungere dei dati in fondo ad un file con un comando del tipo di:

```
cat file addendum > file
```

si potrebbe andrebbe incontro ad una sgradita sorpresa.

Quello che avviene con la redirezione infatti è che se un file esiste questo prima viene "*troncato*",[16] e poi ci viene scritto sopra. Il che comporta che nell'esempio in questione `cat`, alla lettura di `file`, lo troverebbe vuoto. Nell'esempio citato `cat` è in grado di rilevare la situazione anomala e stampare un errore, ma non è detto che altri programmi (o altre versioni di `cat`) siano in grado di farlo; se ad esempio si fosse usato `less` (che usato con la redirezione si comporta esattamente come `cat`) si sarebbe perso il contenuto originale di `file`.

Per ovviare a questo problema è disponibile un altro operatore di redirezione, "`>>`", che redirige lo *standard output* su di un file eseguendo però la scrittura in "*append*", una modalità di scrittura speciale in cui i dati vengono aggiunti in coda a quelli già presenti. In questo modo si può realizzare l'aggiunta di informazioni alla fine del file `file` con il comando:

```
cat addendum >> file
```

Si tenga presente che avere eseguito la redirezione dello *standard output* non comporta il fatto che un programma a questo punto non scriva più nulla sul terminale, si ricordi infatti che gli errori vengono scritti sullo *standard error*, che essendo un file diverso non verrà rediretto; se pertanto si verificano errori nell'esecuzione di un comando i relativi messaggi continueranno ad essere stampati sul terminale. Dato che anche i messaggi di errore possono essere interessanti, si potrà effettuare la redirezione anche per lo *standard error* utilizzando l'operatore "`2>`", così per salvare gli errori di un comando si dovrà usare qualcosa del tipo:

```
comando 2> errori.txt
```

si tenga presente inoltre che la sintassi dell'uso del numero di file descriptor prima dell'operatore funziona anche per gli altri operatori di redirezione, questo ad esempio significa che si potrà anche effettuare la redirezione in *append* dello *standard error* utilizzando come operatore "`2>>`".

Se poi si vuole eseguire la redirezione sia dello *standard output* che *standard error* basterà redirigerli entrambi. Fintanto che si utilizzano due file diversi basterà semplicemente utilizzare gli operatori visti finora specificati in un ordine qualsiasi, si presenta però un problema qualora si volesse effettuare la redirezione di entrambi sullo stesso file, perché usando due redirezioni distinte questo verrebbe scritto attraverso due file descriptor indipendenti.[17] Per questo motivo

[16] esiste una apposita funzione di sistema (per i dettagli si può consultare sez. 4.3.3 di [GaPiL]) che consente di *accorciare* a piacere la dimensione di un file; per evitare sovrascritture parziali di contenuti già esistenti effettuando una redirezione il file di destinazione viene preventivamente troncato a lunghezza nulla.

[17] il risultato porterebbe a delle sovrascritture, la problematica è piuttosto sottile e se ne possono afferrare tutte le conseguenze solo con una conoscenza dettagliata delle modalità con cui vengono gestiti file e file descriptor in un sistema unix-like, l'argomento va al di là di quanto sia possibile affrontare in questo testo, chi fosse interessato può consultare il capitolo 5 di [GaPiL].

viene fornito un operatore diverso, "`&>`", che permette di effettuare questa operazione, così se si vogliono scrivere sia i risultati che gli errori di un comando sullo stesso file si potrà usare qualcosa del tipo:

```
comando &> output.txt
```

esistono comunque altre sintassi equivalenti per effettuare la stessa operazione, riportate in tab. 2.8.

La redirezione di ingresso ed uscita, per quanto utile, è però di gran lunga meno significativa rispetto ad una terza forma di redirezione, detta *pipelining*, che viene realizzata tramite l'operatore "`|`". È questa funzionalità infatti quella che permette di eseguire quella concatenazione dei comandi a cui abbiamo accennato in sez. 2.1.1.

L'operatore "`|`" consente di collegare lo *standard output* del comando che lo precede con lo *standard input* del comando che lo segue. Questo viene fatto utilizzando una "*pipe*", un servizio di comunicazione fra processi fornito dal kernel che, come indica il nome stesso, costituisce una sorta di *conduttura* di collegamento fra i due comandi.

Una *pipe* è del tutto analoga alle *fifo* incontrate in sez. 1.2.1, quello che un processo scrive su un capo, viene letto da un altro processo sull'altro capo, solo che una *pipe* non è associata ad un oggetto sul filesystem, ma esiste solo come file descriptor accessibile dall'interno dei processi interessati. Se questi file descriptor sono rispettivamente lo *standard output* di un primo comando e lo *standard input* del secondo si otterrà il collegamento voluto e sarà possibile fornire come dati in ingresso ad un comando quelli prodotti in uscita da un altro.

Si possono così lanciare in sequenza una serie di comandi in cui ciascuno elabora i risultati del precedente, effettuando combinazioni ed elaborazioni anche molto complesse. Come esempio proviamo a contare in quanti file di testo è suddiviso il contenuto di questo testo, se ricordiamo che l'opzione `-n` del comando `cat` consente di numerare le righe di un file, potremo ottenere una lista numerata con:

```
piccardi@ellington:~/truedoc/corso$ ls *.tex | cat -n
     1	advadmin.tex
     2	appendici.tex
     3	config.tex
     4	corso.tex
     5	netadmin.tex
     6	netbase.tex
     7	netdns.tex
     8	netinter.tex
     9	netlpi.tex
    10	ordadmin.tex
    11	ringraziamenti.tex
    12	shell.tex
    13	stradmin.tex
    14	struttura.tex
```

mandando la lista dei file generata da `ls` in ingresso a `cat`.

La disponibilità di questa funzionalità comincia a farci intuire quali possono essere le capacità intrinseche della riga di comando. Ad esempio non è più necessario dover inserire in tutti i comandi una opzione per ordinare in una qualche maniera i risultati prodotti in uscita, basterà inviare questi ultimi con una *pipe* al programma `sort`, il cui solo scopo è quello di effettuare riordinamenti nelle modalità più varie.

<table>
<tr><th>Comando</th><th>Significato</th></tr>
<tr><td><code>cmd < file</code></td><td>redirige lo standard input: legge l'input del comando <code>cmd</code> dal file <code>file</code>.</td></tr>
<tr><td><code>cmd > file</code></td><td>redirige lo standard output: scrive il risultato del comando <code>cmd</code> sul file <code>file</code>.</td></tr>
<tr><td><code>cmd >> file</code></td><td>redirige lo standard output: scrive il risultato del comando <code>cmd</code> accodandolo al contenuto del file <code>file</code>.</td></tr>
<tr><td><code>cmd 2> file</code></td><td>redirige lo standard error: scrive gli errori del comando <code>cmd</code> sul file <code>file</code>.</td></tr>
<tr><td><code>cmd1 | cmd2</code></td><td>pipelining: redirige lo standard output del comando <code>cmd1</code> sullo standard input del comando <code>cmd2</code>.</td></tr>
<tr><td><code>cmd > file 2>&1</code></td><td>redirige lo standard output e lo standard error: scrive errori e risultati del comando <code>cmd</code> sul file <code>file</code>.</td></tr>
<tr><td><code>cmd >& file</code></td><td>stesso effetto del precedente.</td></tr>
<tr><td><code>cmd &> file</code></td><td>stesso effetto del precedente.</td></tr>
<tr><td><code>cmd1 2>&1 | cmd2</code></td><td>redirige lo standard output e lo standard error del comando <code>cmd1</code> sullo standard input del comando <code>cmd2</code>.</td></tr>
</table>

Tabella 2.8: Principali modalità di redirezione.

Vedremo in sez. 2.2 molti altri esempi di come il *pipelining* può essere sfruttato per costruire delle vere e proprie *catene di montaggio* in cui si ottiene un risultato finale attraverso la concatenazione di molti comandi, concludiamo questa sezione riportando in tab. 2.8 un riassunto delle principali modalità di redirezione utilizzate nella shell, comprese alcune delle più esoteriche che permettono di redirigere più file; per una trattazione completa si consulti la pagina di manuale, alla sezione "`REDIRECTION`".

2.1.5 Scripting elementare

Una delle capacità fondamentali della shell è che si possono inserire, come avveniva per i file `.bat` del DOS, delle sequenze di comandi in un file per farli eseguire direttamente senza doverli riscrivere tutte le volte. Ma se questo è più o meno tutto quello che si può fare con il DOS, il grande vantaggio di una shell Unix è che si ha a disposizione un vero e proprio linguaggio di programmazione con tanto di variabili e direttive di iterazione, condizionali, ecc. che permette di effettuare anche compiti di notevole complessità. E nonostante che come linguaggio di programmazione sia poco elegante e piuttosto limitato, lo *shell scripting* viene comunque utilizzato ampiamente, data la sua immediatezza, per automatizzare una gran quantità di operazioni per le quali non vale la pena ricorrere ad un linguaggio di più alto livello.

È per questo motivo che la shell, anche se finora ne abbiamo parlato solo come del programma che implementa l'interfaccia a riga di comando, è in realtà uno dei componenti essenziali per far funzionare il sistema, si noti infatti che, come richiesto dal *Filesystem Hierarchy Standard* trattato in sez. 1.2.3, essa deve trovarsi sotto `/bin`, dato che è essenziale per l'avvio del sistema. Questo perché non è necessaria solo dal punto di vista dell'interfaccia utente: anche i meccanismi di avvio del sistema infatti, come vedremo in sez. 5.3.5, usano proprio degli *script di shell* per eseguire le loro operazioni.

La descrizione dettagliata della shell come linguaggio di programmazione va al di là dello scopo di questo testo, le sue caratteristiche principali si trovano comunque nella sezione "`SHELL GRAMMAR`" della pagina di manuale, ed un ottima risorsa per quanti vogliano approfondirlo è

[AdvBash]. Quello che è importante sottolineare qui è che la shell non solo può essere invocata in maniera interattiva, quando è associata ad un terminale ed utilizzata dagli utenti per dare comandi, ma può anche essere lanciata non interattivamente facendole eseguire direttamente quello che appunto si chiama uno *script*. In questo caso la si invoca come un qualunque altro comando, e prende come argomento il file dello script da eseguire, con una sintassi del tipo `bash script`.

In realtà gli script possono sfruttare una funzionalità peculiare del cosiddetto *link-loader* di Linux (che vedremo anche in sez. 3.1.2) comune a tutti i sistemi unix-like, che oltre ad eseguire i programmi nel formato binario standard è in grado di eseguire anche gli script, che vengono trattati esattamente allo stesso modo. Questi sono identificati dal *link-loader* come file di testo dotati di permesso di esecuzione la cui prima riga è nella forma:

```
#!/bin/bash
```

in cui cioè si trovano i caratteri "`#!`" seguiti dal pathname di un programma che faccia da interprete. In tal caso il *link-loader* lancerà automaticamente il programma interprete (con le opzioni che si possono specificare sempre sulla prima riga del file) dandogli come argomento il nome file stesso, così che questo esegua le istruzioni in esso contenute.

Dato che per la shell e per tutti i linguaggi di scripting che usano questa funzionalità, come il Python o il Perl, il carattere "`#`" indica l'inizio di una riga di commento, quando si vuole creare uno script basterà scrivere un file di testo contenente tutte le istruzioni necessarie che inizi con questa riga, che verrà ignorata nell'interpretazione dei comandi, e poi renderlo eseguibile.

Si tratta cioè di attivarne il permesso di esecuzione con `chmod +x` anche se poi dal punto di vista del contenuto del file non cambierà assolutamente nulla, restando questo un normale file di testo. Ma in questa maniera uno script sarà del tutto equivalente ad un comando binario, ed oltre ad eseguirlo come tale la shell provvederà anche a passargli gli argomenti scritti sulla riga di comando, esattamente come avrebbe fatto se si fosse trattato di un eseguibile compilato.

Oltre alla sintassi di base illustrata in sez. 2.1.3, che resta sempre disponibile, quando si scrivono degli script di shell si fa spesso ricorso a istruzioni e costrutti sintattici il cui uso è poco comune sulla riga di comando, sia perché essi sono attinenti a funzionalità più complesse che sono poco indicate per un uso interattivo, sia perché dipendono da funzionalità direttamente legate all'uso all'interno degli script.

Una sintassi poco utilizzata nell'uso interattivo è quella che permette di eseguire più comandi sulla stessa linea; cosa che può essere fatta in tre modalità diverse. La prima modalità prevede che i singoli comandi siano separati fra di loro tramite il carattere "`;`". In tal caso si tratta semplicemente di scrivere su una riga unica quello che altrimenti si scriverebbe su linee diverse, usando il "`;`" per separare la fine di un comando dall'inizio del successivo. In questo caso i vari comandi saranno eseguiti in sequenza da sinistra a destra come se al posto del "`;`" si fosse premuto invio.

Questa sintassi è di scarsa utilità sulla linea di comando, se non per eseguire più comandi in sequenza, senza dover stare ad attendere che ciascuno di essi finisca per poter scrivere il successivo. Ha però una sua utilità specifica negli script dove alcune direttive di programmazione, come i condizionali che vedremo a breve, usano come parametro una linea di comando, la cui terminazione può essere resa esplicita con questo metodo.

Una seconda modalità di esecuzione multipla è quella in cui i comandi che si vogliono eseguire vengono separati dall'operatore "`&&`", che assume il significato di AND logico. In questo caso

il comando successivo all'operatore verrà eseguito soltanto se il precedente non ha avuto errori mentre in caso di errori la catena verrà immediatamente interrotta. Questo operatore consente di eseguire un secondo comando solo se il primo ha avuto successo e l'uso di questa sintassi è comune negli script ad esempio per verificare se un certo file esiste e in caso affermativo eseguire un comando.

Una terza modalità di esecuzione multipla prevede che i diversi comandi sulla stessa linea vengano separati dall'operatore "`||`", che assume il significato di OR logico. In questo caso si avrà che il comando successivo all'operatore verrà eseguito soltanto se il precedente ha fallito. Questo consente di eseguire in alternativa due comandi; in genere lo si usa negli script per eseguire le operazioni da effettuare solo in caso di errori.

Queste due ultime modalità fanno riferimento, per stabilire se un comando ha avuto o meno successo, a quella caratteristica comune di tutti i programmi che una volta terminati riportano al programma che li ha lanciati (nel caso specifico la shell) uno stato di uscita che serve ad indicare se le operazioni sono state concluse correttamente. Abbiamo già accennato a questa funzionalità in sez. 1.3.2, quando abbiamo trattato degli *zombie* che derivano dal non ricevere correttamente questo stato di uscita. La shell è ovviamente programmata per riceverlo correttamente, e ne utilizza il valore proprio per determinare se il programma è finito correttamente o meno.

La convenzione utilizzata universalmente è che un valore nullo (cioè 0) significa il successo delle operazioni, mentre un valore non nullo (in genere 1) indica che c'è stato un errore. Inoltre se il programma non si è concluso spontaneamente ma è stato interrotto da un segnale la shell assegna al codice di uscita il valore di $128 + n$, dove n è il numero del segnale (ed ovviamente il comando viene considerato fallito).

Tutte le volte che un programma viene eseguito la shell riceve questa informazione, ed è con un controllo automatico su questi valori che, nell'uso degli operatori "`&&`" e "`||`", vengono prese le decisioni sull'esecuzione o meno dei vari comandi che la shell concatena tramite essi.

La presenza di un codice di uscita è molto importante, specie per gli script, che possono usare questa informazione per capire se ci sono stati problemi nell'esecuzione di un comando. Per questo la shell, nel ricevere lo stato di uscita di un comando, lo memorizza nella variabile speciale "`?`". In questo modo nella riga di comando successiva o in uno script diventa possibile controllare lo stato di uscita dell'ultimo comando eseguito facendo riferimento ad essa con "`$?`".

Si deve inoltre precisare che tutte le volte che si concatenano più comandi, sia con gli operatori condizionali appena visti, che con una *pipe*, che con l'uso del "`;`", alla fine dell'esecuzione della intera riga di comando nella variabile "`?`" verrà riportato il codice di uscita dell'ultimo programma eseguito.

Vediamo allora un esempio di come utilizzare questa funzionalità, potremo verificare che in un caso in cui il comando viene eseguito senza errori si ottiene:

```
piccardi@anarres:~/truedoc$ ls
Makefile       adminbook.tex  images                 macro.aux    sicurezza
README         commons        introduzione_sviluppo  mailservices webservices
...
piccardi@anarres:~/truedoc$ echo $?
0
```

mentre qualora si abbia un errore si ottiene:

```
piccardi@anarres:~/truedoc$ ls licidi
ls: cannot access licidi: No such file or directory
```

```
piccardi@anarres:~/truedoc$ echo $?
1
```

quindi nel primo caso, in cui il comando è stato eseguito correttamente, il valore della variabile "?" è 0, mentre nel secondo, in cui `licidi` non esiste, si ha un errore ed il valore è 1.

La variabile "?" è una delle *variabili speciali* della shell descritte nella sezione "`Special Parameters`" della pagina di manuale. Queste variabili vengono mantenute direttamente dalla shell stessa (non ha cioè senso assegnarle direttamente come visto in sez. 2.1.3) che vi memorizza il valore di alcuni parametri interni. L'elenco delle principali *variabili speciali* è riportato in tab. 2.9.

Variabile	**Contenuto**
`#`	il numero degli argomenti passati ad uno script.
`*`	la lista degli argomenti. Se espansa fra virgolette come `"$*"` è equivalente ad una unica stringa formata dal contenuto dei singoli argomenti separato da spazi (o dal carattere di separazione specificato in `IFS`).
`@`	la lista degli argomenti. Se espansa fra virgolette come `"$@"` è equivalente alla lista dei singoli argomenti.
`?`	il codice di uscita dell'ultimo comando eseguito.
`$`	il PID della shell.
`!`	il PID del programma in *background* che è stato eseguito per ultimo.
`0`	il nome della shell o dello script.
`-`	la lista delle opzioni con cui è stata invocata o che sono attive per la shell corrente.

Tabella 2.9: Le principali *variabili speciali* della shell.

Alcune di queste *variabili speciali* sono ad uso specifico degli script; ad esempio quando si lancia uno script con degli argomenti con `$#` si ottiene il numero degli stessi, con `$1` il primo argomento, con `$2` il secondo, ecc. È inoltre possibile ottenere una stringa contenente tutti gli argomenti con `$*`, mentre con `$@` si ottiene una lista.

La differenza fra `$*` e `$@` può sembrare non significativa, e se si stampano le due variabili con `echo` non se ne vedrà nessuna. In realtà questa è fondamentale negli script, infatti `$*` nelle istruzioni usate negli script viene considerata come un'unica stringa, mentre `$@` viene espansa in una lista, costituita da tanti elementi quanti sono gli argomenti passati allo stesso. Questo comporta ad esempio che nel caso si usi la prima in una istruzione `for` si avrà un solo ciclo, mentre con la seconda si avranno tanti cicli quanti sono gli argomenti.

Per capire meglio come si usano facciamo un esempio, consideriamo allora il seguente script:

`prova.sh`

```
#! /bin/sh
echo "Questo script ha $# argomenti"
echo "Argomento n.1 $1, n.2 $2, ecc."
echo "l'insieme degli argomenti: $@"
```

in cui si utilizzano le variabili speciali relative al numero di argomenti, al primo ed al secondo argomento ed alla lista completa; se lo si esegue passandogli tre argomenti si otterrà come risultato:

```
piccardi@ellington:~$ ./prova.sh arg1 arg2 arg3
Questo script ha 3 argomenti
Argomento n.1 arg1, n.2 arg2, ecc.
l'insieme degli argomenti: arg1 arg2 arg3
```

Oltre a ricevere degli argomenti sulla riga di comando, uno script di shell può anche, come i normali programmi, leggere dati dallo *standard input* ed interagire anche in questo modo con l'utente. Il comando più usato per ottenere dati in questo modo è `read`, che legge una riga dallo *standard input* e la divide in parole che assegna come valore alle variabili che si sono specificate come argomento.

In questo caso di nuovo viene usata la variabile `IFS` per suddividere in parole la riga. Si tenga presente che il comando legge una riga alla volta, per cui se dobbiamo specificarne più di una occorrerà rieseguire il comando o fare un ciclo. Inoltre il comando esegue l'assegnazione delle singole parole alle singole variabili fintanto che queste sono nello stesso numero. Se si sono indicate più variabili che parole, alle variabili eccedenti sarà assegnato un valore nullo, viceversa se ci sono più parole che variabili all'ultima variabile sarà assegnata la parte di riga restante.

Oltre alla possibilità di passare argomenti, e di leggere dati dallo *standard input*, un'altra delle caratteristiche che rendono uno script di shell qualcosa di molto più sofisticato di una semplice sequenza di righe di comando, ed affine un programma vero e proprio, è il supporto per una serie di *istruzioni* che consentono di creare costrutti sintattici complessi, propri di un linguaggio di programmazione. Di questi illustreremo solo i più significativi, come accennato l'elenco completo è nella sezione "`SHELL GRAMMAR`" della pagina di manuale.

Abbiamo visto come i due operatori "`&&`" e "`||`" consentano di effettuare un'esecuzione condizionata di un comando a seconda dello stato di uscita del precedente. La shell consente però anche una costruzione più esplicita e completa delle esecuzioni condizionali, basandosi sull'uso dell'istruzione `if`, la cui forma generica è:

```
if condition_cmd;  then
   command
   ...
else
   command
   ...
fi
```

In questo caso in genere non si usa una singola riga di comando in quanto la sintassi richiesta prevede l'uso di più istruzioni, combinate opportunamente. In particolare l'esempio precedente ci mostra come dopo l'`if` iniziale si debba scrivere il comando il cui stato di uscita viene a costituire la condizione da controllare, seguito dall'istruzione `then`. Si noti anche come nell'esempio si sia indicata esplicitamente la fine del comando con il carattere "`;`" senza il quale sarebbe stato necessario scrivere l'istruzione `then` su una riga a parte.

Il comando soggetto dell'istruzione `if` viene sempre eseguito ed a seconda del suo risultato verranno eseguiti la lista di comandi inseriti dopo l'istruzione `then` se questo ha successo (cioè riporta un valore di uscita nullo) o quelli scritti dopo l'istruzione `else` se questo fallisce (cioè riporta un valore di uscita non nullo). Il blocco condizionale deve comunque essere sempre terminato esplicitamente con l'istruzione `fi`. L'uso di `else` è opzionale, se non ci sono alternative si possono specificare solo i comandi da eseguire se il comando ha successo fra `then` e `fi`.

La precedente sintassi può essere resa ancora più complessa in quanto è possibile realizzare blocchi condizionali annidati sostituendo l'istruzione `else` con l'istruzione `elif`. L'istruzione è sostanzialmente identica ad `if`, e consente di eseguire come alternativa, al posto dei comandi posti dopo l'istruzione `else`, un ulteriore blocco condizionale, con la stessa sintassi appena illustrata, introdotto dall'`elif` invece che da un `if`.

Oltre all'esecuzione condizionata di blocchi di istruzioni, la shell permette di eseguire cicli condizionali tramite l'uso delle due istruzioni `while` e `until`. Entrambe le istruzioni permettono la ripetizione di un insieme di comandi sulla base del risultato di un altro comando (specificato come argomento) che serve ad esprimere la condizione. La lista dei comandi da eseguire nel ciclo deve essere delimitata dalle istruzioni `do` e `done`.

Il comando che esprime la condizione viene sempre valutato all'inizio del ciclo; le successive istruzioni vengono eseguite se ha successo nel caso di `while` o se fallisce nel caso di `until`. Se il ciclo condizionale viene eseguito, alla sua conclusione la condizione viene rivalutata con la riesecuzione del comando iniziale, ed il ciclo eventualmente ripetuto a seconda del nuovo risultato ottenuto. Un esempio della sintassi per `while` è il seguente:

```
while condition_cmd; do
    command
    ...
done
```

e la sintassi è identica nel caso si usi `until`.

Se si devono operare scelte diverse sulla base del valore di una variabile si può usare l'istruzione `case`. In generale `case` prende come argomento una espressione che possa assumere valori diversi. Il caso più comune è quello di una variabile da referenziare con la *variable expansion*, ma si può utilizzare anche il risultato di una *command expansion*. L'istruzione permette di eseguire comandi diversi a seconda dei valori che l'espressione di controllo può assumere, con una sintassi generica del tipo di:

```
case expression in
    value1)
        command
        ...
    ;;
    value2)
        command
        ...
    ;;
    *)
        command
        ...
    ;;
esac
```

in cui ogni caso viene terminato da un doppio "`;;`" ed i singoli valori con cui viene confrontata l'espressione sono terminati da una parentesi chiusa. Si può esprimere il valore generico con un "`*`". La lista dei casi deve sempre essere terminata con l'istruzione `esac`.

L'utilità di `case` rispetto all'uso di una serie concatenata di `if` ed `elif`, oltre alla leggibilità, è che i valori con cui effettuare il controllo supportano i caratteri jolly con lo stesso significato che questi assumono nel *filename globbing* visto in sez. 2.1.3, anche se in questo caso vengono applicati alla lista dei valori forniti dall'espressione piuttosto che ai nomi dei file, (è per questo che si può usare "*" per indicare il valore generico). Questa funzionalità consente di raggruppare in maniera flessibile valori simili e trattarli con una unica serie di istruzioni.

Infine l'istruzione `for` permette di compiere dei cicli su un numero stabilito di alternative, definendo al contempo una variabile che può essere utilizzata all'interno del ciclo, con una sintassi del tipo:

```
for var in list ; do
   command
   ...
done
```

dove di nuovo le istruzioni `do` e `done` delimitano le istruzioni compiute all'interno del ciclo mentre `var` è la variabile che assume, per ciascuna ripetizione del ciclo, i vari possibili valori risultanti dall'espansione dell'espressione `list`.

Il comando interpreta semplicemente il valore di `list` come una lista costituita di valori singoli separati da spazi, assegnando in sequenza i singoli valori a `var` per ciascuna ripetizione del ciclo. La potenza del comando risiede nel fatto che per esprimere `list`, oltre ad una eventuale elencazione esplicita dei valori, si può utilizzare sia la *variable expansion*, che la *command expansion* che il *filename globbing*.

Come complemento alla possibilità di eseguire ciclicamente un gruppo di comandi con istruzioni come `for`, `while` ed `until`, sono state previste due istruzioni che permettono di controllare il flusso di esecuzione all'interno di un ciclo. La prima istruzione è `continue`, che conclude immediatamente il ciclo corrente per passare immediatamente al successivo, saltando tutte le istruzioni restanti. La seconda è `break`, che termina immediatamente la ripetizione del ciclo e passa alla prima istruzione ad esso seguente.

Siccome è possibile annidare più cicli uno dentro l'altro, entrambe le istruzioni supportano un argomento numerico che indica di quanti livelli tornare indietro, il valore deve essere un intero maggiore di uno, e se si specifica un livello superiore al numero di annidamenti si tornerà semplicemente al più esterno dei cicli presenti.

Una delle caratteristiche comuni a tutti i linguaggi di programmazione, e la shell non fa eccezione, è quella della possibilità di definire delle funzioni da richiamare per eseguire direttamente in diverse occasioni la stessa serie di istruzioni. Questa capacità ha una conseguenza sulla sintassi della shell, perché quando si scrive un comando questo può fare riferimento, oltre ad eventuali programmi o script o ad un alias per un altro comando, ad una funzione, che viene appunto eseguita scrivendone il nome.

Benché sia possibile farlo anche sulla riga di comando, in generale una funzione è definita all'interno di uno script, e viene introdotta dalla parola chiave `function` cui segue il nome della funzione e il costrutto "`()`". In realtà l'istruzione `function` è opzionale e può essere omessa, perché per la definizione di una funzione è sufficiente la presenza di un nome seguito dal costrutto "`()`", quest'ultimo a sua volta può essere omesso se si usa l'istruzione `function`.

Dopo di che il corpo della funzione, che contiene tutte le istruzioni da essa eseguite, deve essere scritto di seguito delimitato fra parentesi graffe. In sostanza la forma di una funzione di shell è sempre qualcosa del tipo:

```
function name () {
   command
   ...
}
```

dove ciascuna linea conterrà un comando o altre istruzioni, esattamente come si potrebbero mettere in uno script; a differenza di uno script però queste verranno eseguite nella shell corrente e non in un processo a parte.

L'uso degli script ci pone di fronte anche alcune conseguenze non immediate del funzionamento dell'architettura dei processi. Come accennato infatti quando si lancia uno script di shell esso viene eseguito come un comando: la shell crea un nuovo processo figlio in cui esegue quella che si chiama una *subshell*, cioè un'altra istanza di se stessa che interpreta lo script. Tutto questo ha delle precise conseguenze: ad esempio se in uno script si effettuano modifiche alle variabili di ambiente o agli *alias* queste saranno effettive solo per la relativa *subshell*, e non per la shell originaria che l'ha lanciato, non essendo possibile ad un processo modificare le proprietà di un altro processo.

Per questo motivo, se si vogliono automatizzare una serie di impostazioni per la shell corrente, ad esempio impostare alcune variabili di ambiente, non si può pensare di farlo con uno script eseguito in una *subshell* che contenga i relativi comandi. È però possibile far eseguire direttamente alla shell corrente i comandi scritti in un file, senza usare una *subshell*, ottenendo così le impostazioni volute.

Per fare questo è disponibile il comando interno `source` che richiede il nome del file da cui leggere le istruzioni come argomento. Una sintassi alternativa, che però è quella che è stata standardizzata da POSIX.2 e si trova pertanto su tutte le shell compatibili, è quella che prevede l'uso di "`.`" al posto di `source`. In questo caso non è necessario che il file contenente i comandi sia eseguibile e presenti la sintassi di uno script con il "`#!`" iniziale, dato che il suo contenuto viene semplicemente letto ed eseguito dalla shell come se lo si fosse scritto direttamente sul terminale.

La `bash` inoltre è in grado di distinguere questa condizione ed accorgersi se viene eseguita o meno in una *subshell* tramite il valore della variabile predefinita `SHLVL` che indica il cosiddetto "*livello*". Questo valore parte da 1 per le shell lanciate direttamente da un altro programma, come quella ottenuta all'avvio o al lancio di un terminale virtuale, e viene incrementato di uno ogni volta che una shell viene lanciata da un'altra shell. In questo modo è possibile capire se si è all'interno di una *subshell* o meno ed eventualmente tenerne conto all'interno degli script.

Un comando interno complementare a `source` è `exec` che consente di eseguire un altro programma al posto della shell corrente. In questo caso quello che succede è che il processo resta lo stesso (e ad esempio non cambia il suo PID), ma viene messo in esecuzione al suo interno il codice del programma passato come argomento ad `exec`. Questo programma può anche essere uno script, ma si deve tenere presente che in tal caso, pur restando il processo lo stesso, il programma messo in esecuzione sarà la *subshell* che interpreta lo script, che verrà eseguita al posto della shell corrente, sostituendola completamente. Questo significa che a differenza di quanto avviene con l'uso di `source` si può lanciare con `exec` solo uno script eseguibile e non un semplice

file, e che alla terminazione dello script si avrà anche la terminazione del processo, senza poter tornare nella shell da cui si era partiti.

Infatti con `exec` la conclusione del programma che si è fatto eseguire fa terminare anche il processo da cui si è partiti, e se questo era la shell con cui ci si è collegati al sistema, si ottiene anche la conclusione della sessione di lavoro. Per questo in genere questa istruzione viene utilizzata all'interno degli script quando questi devono porre in esecuzione al proprio posto altri programmi.

Oltre alle istruzioni interne della shell esistono anche una serie di comandi di utilità che sono stati creati proprio per facilitare la scrittura di script. Il primo di questi è `seq`, che permette di stampare sullo *standard output* una sequenza di numeri.

Il comando può essere invocato con un numero di argomenti variabile da uno a tre numeri interi: un solo argomento indica il numero finale a cui fermare la sequenza, partendo da 1 ed incrementando di uno, specificando due argomenti si indicano il valore iniziale ed il valore finale, che viene raggiunto sempre a passi di uno. Infine se si forniscono tre argomenti il secondo argomento viene usato come valore di incremento nell'intervallo che inizia con il primo e finisce col terzo. Questo comando è molto utile quando si vogliono fare dei cicli, lo si può infatti usare per generare una lista di numeri da utilizzare con l'istruzione `for`, i cui valori possono poi essere utilizzati all'interno del ciclo; ad esempio per eseguire dieci volte la stessa serie di comandi si potrà usare qualcosa del tipo:

```
for i in `seq 1 10`; do
   command
   ...
done
```

Altri due comandi speciali sono `true` e `false`, che non fanno assolutamente nulla se non fornire alla shell un valore di ritorno. Nel caso di `true` si avrà un valore nullo, corrispondente ad una esecuzione sempre corretta, che nelle istruzioni condizionali implica vero. Se invece si usa `false` questo restituisce un valore non nullo, cioè un fallimento del comando, equivalente alla condizione falsa. Questi due programmi vengono usati spesso come shell di default (vedi sez. 4.3.2) per gli account degli utenti di sistema che non devono avere una shell effettiva.

Un altro comando utilizzato negli script per il suo codice di uscita è `test` che è uno dei comandi più comuni utilizzato come argomento per le istruzioni condizionali illustrate in precedenza, e che nel caso di `bash` è presente anche come comando interno. Dato che il comando serve ad eseguire dei controlli su cui esprimere una condizione, una modalità alternativa per invocarlo è con il nome "`[`" (si può infatti osservare l'esistenza del file `/usr/bin/[`) nel qual caso la linea di comando che lo utilizza dovrà essere terminata con una corrispondente "`]`".

È questo il motivo per cui le condizioni che si trovano spesso negli script di shell richiedono che dopo la parentesi quadra sia sempre presente uno spazio: non si tratta infatti di una sintassi interna della shell attivata da un carattere di controllo, ma di un comando vero e proprio, che come tale deve essere riconosciuto con le regole classiche dei nomi dei comandi, che sono separati da spazi dai relativi argomenti.

Il comando può eseguire controlli su un gran numero di condizioni diverse, passate come argomenti in forma di una opportuna espressione, ritornando un valore nullo se la condizione è verificata, e non nullo altrimenti. Il comando inoltre supporta l'uso di alcuni *operatori*, come "`!`", che anteposto ad una espressione ne nega il risultato, e la combinazione logica di due espressioni

con l'uso delle opzioni `-a` e `-o`, che indicano rispettivamente l'AND e l'OR. In questo modo si possono costruire delle condizioni molto complesse.

Opzione	Significato
`-b` *file*	il file esiste ed è un dispositivo a blocchi.
`-c` *file*	il file esiste ed è un dispositivo a caratteri.
`-d` *file*	il file esiste ed è una directory.
`-e` *file*	il file esiste.
`-f` *file*	il file esiste ed è un file normale.
`-h` *file*	il file esiste ed è un link simbolico.
`-O` *file*	il file esiste ed è di proprietà dell'*effective user ID* del processo.
`-G` *file*	il file esiste ed è proprietà dell'*effective group ID* del processo.
`-r` *file*	il file esiste ed è leggibile.
`-s` *file*	il file esiste ed ha dimensione maggiore di zero.
`-w` *file*	il file esiste ed è scrivibile.
`-x` *file*	il file esiste ed è eseguibile.

Tabella 2.10: Opzioni per le condizioni sui file del comando `test`.

Buona parte delle condizioni verificate da `test` fanno riferimento alle proprietà dei file, in questo caso la condizione viene espressa con una opzione seguita dal nome del file cui la si vuole applicare. Le principali opzioni utilizzate, ed il relativo significato, sono riportati in tab. 2.10.

Oltre alle condizioni sui singoli file il comando può essere usato per esprimere condizioni relative a due file, in tal caso queste vengono espresse scrivendo questi ultimi come argomenti separati dalla relativa opzione. Con `-ef` si richiede che essi abbiano lo stesso *inode*, con `-nt` si richiede che il primo sia più recente (in termini di tempo di ultima modifica, vedi sez. 1.2.1) mentre con `-ot` che sia più vecchio (negli stessi termini).

Una ultima serie di condizioni riguarda l'uso di stringhe e numeri; per questi ultimi valgono gli operatori di confronto dei valori come `-eq`, `-ge`, `-gt`, `-le`, `-lt` e `-ne`, il cui significato è banale (uguale, maggiore uguale, maggiore, minore uguale, minore, e diverso). Per le stringhe invece si possono usare gli operatori "`=`" e "`!=`" per indicare rispettivamente uguaglianza e disuguaglianza, mentre l'opzione `-z` serve a verificare se la stringa è vuota e l'opzione `-n` se non lo è.

2.1.6 Le modalità operative e la configurazione della shell

L'esempio degli script rende evidente come, benché il programma sia lo stesso, la shell possa essere utilizzata in modi diversi. In generale una shell usata per eseguire uno script necessiterà di impostazioni diverse rispetto ad una shell usata per gestire la riga di comando (ad esempio non serve il *prompt*). Per poter effettuare delle impostazioni diverse a seconda del tipo di utilizzo la `bash` supporta modalità di funzionamento che vengono abilitate a seconda di come essa viene lanciata, si ricordi infatti che la shell è un programma come gli altri e la si può eseguire direttamente sulla riga di comando esattamente come gli altri, ottenendo appunto una *subshell*.

In particolare quando ci si collega ad una macchina attraverso la shell, che sia su una *console*, con un emulatore di terminale o tramite una connessione via rete, devono essere eseguite una serie di inizializzazioni, come la definizione del prompt e della variabile `PATH`, che normalmente non servono nel caso di uno script, che eredita le impostazioni dalla shell corrente. Per questo esiste una apposita modalità di invocazione che consente di ottenere quella che viene chiamata una *shell di login*, che può essere attivata esplicitamente dall'uso dell'opzione `-l`.

In realtà i vari programmi che lanciano una *shell di login* (come `login` per l'accesso da *console*) invece di usare l'opzione `-l`, lo fanno apponendo un "`-`" come primo carattere dell'argomento zero (quello che indica il nome del programma) fra quelli passati alla shell posta in esecuzione, il che poi ci permette di identificare le shell di login nell'output di `ps`, come si può vedere anche nell'esempio di pag. 27. Dato che la `bash` analizza gli argomenti, compreso questo, se riconosce di essere stata lanciata come `-bash` provvederà a comportarsi come *shell di login*.

Quello che contraddistingue una shell di login è che al suo avvio vengono sempre eseguite una serie di procedure di inizializzazione specifiche. La prima di queste è l'esecuzione all'interno della shell stessa, come se si fosse usato il comando `source`, di tutte le istruzioni contenute nel file `/etc/profile`, in cui l'amministratore di sistema può inserire una serie di impostazioni comuni per tutti gli utenti. Questa è una caratteristica generale che hanno tutte le shell derivate dalla Bourne shell, e non solo la `bash`,[18] pertanto occorre, come nell'esempio, tenerlo presente ed utilizzare funzionalità specifiche della `bash` solo in maniera condizionale.

Un esempio di questo file, che può essere considerato una sorta di file di configurazione della shell, è riportato di seguito. L'esempio è preso dalla versione installata di default su una Debian Squeeze, e la sua lettura può essere considerata anche un buon esercizio per prendere familiarità con alcune delle istruzioni che abbiamo illustrato in sez. 2.1.5:

/etc/profile

```
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ "‘id -u‘" -eq 0 ]; then
  PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
else
  PATH="/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games"
fi
export PATH

if [ "$PS1" ]; then
  if [ "$BASH" ]; then
    # The file bash.bashrc already sets the default PS1.
    # PS1='\h:\w\$ '
    if [ -f /etc/bash.bashrc ]; then
      . /etc/bash.bashrc
    fi
  else
    if [ "‘id -u‘" -eq 0 ]; then
      PS1='# '
    else
      PS1='$ '
    fi
  fi
fi

if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
    if [ -r $i ]; then
      . $i
    fi
  done
```

[18] per chi fosse interessato quelle derivate dalla C shell usano invece `/etc/csh.login`.

```
  unset i
fi
```

Come si può notare nel file si usano dei condizionali per effettuare delle impostazioni, ed in particolare in un paio di occasioni si usa una *command expansion* per controllare il risultato del comando `id` (vedi sez. 2.4.4) che consente di ottenere l'*user ID* dell'utente e decidere quali configurazioni applicare a seconda che questi sia o meno l'amministratore. Si può notare allora come prima venga impostata la variabile `PATH` a seconda del tipo di utente, e poi il prompt con `PS1`, se questa non risulta già definita.

Si noti come anche in questo caso se la shell usata è la `bash`, nel qual caso sarebbe definita la variabile `BASH`, l'impostazione del prompt viene demandata all'uso di `/etc/bash.bashrc` (su cui torneremo a breve) che assegna a `PS1` un valore analogo a quello commentato, che sfrutta le funzionalità illustrate in tab. 2.3. Invece per una shell standard minimale, come potrebbe essere la `dash` usata da Debian per gli script di avvio, non essendo garantite le funzionalità in più della `bash` vengono usati direttamente i caratteri "`#`" e "`$`" a seconda che l'utente sia o meno l'amministratore.

Infine vengono trattate ulteriori eventuali personalizzazioni, da inserire in singoli file sotto `/etc/profile.d/` il cui nome deve terminare in `.sh`. Per questo viene usato un ciclo di `for` sui nomi ottenuti con il *filename globbing*, controllando che i file siano leggibili, e nel caso ne vengono eseguite le istruzioni. In questa versione del file non viene più impostato il valore della *umask* (vedi sez. 1.4.3) per la sessione come avveniva in precedenza, cosa che è demandata ad uno specifico modulo di PAM (tratteremo l'argomento in sez. 4.3.7).

Per permettere agli utenti di aggiungere le loro personalizzazioni alla shell di login, dopo aver letto `/etc/profile` per eseguire le impostazioni comuni a tutto il sistema, la `bash` cerca nella home dell'utente, ed in quest'ordine, i file `.bash_profile`, `.bash_login` e `.profile`, ed esegue i comandi del primo che trova, senza guardare gli altri. Si noti come nessuno di questi file sia eseguibile, dato che non viene eseguito come script, per cui è sufficiente il permesso di lettura. Un esempio di `.profile` (sempre ripreso da una Debian Squeeze) è il seguente:

.profile

```
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
```

```
    PATH="$HOME/bin:$PATH"
fi
```

dove sostanzialmente se la shell è la `bash`, riconosciuta stavolta per la presenza della variabile di default `BASH_VERSION` si utilizzano eventuali impostazioni personali poste nel file `.bashrc` (su cui torneremo a breve) e si aggiunge al `PATH`, se questa esiste, la directory `bin` nella home dell'utente.

Infine all'uscita di una shell di login è prevista l'esecuzione, se questo esiste, delle istruzioni contenute nel file `.bash_logout`, che può essere utilizzato per effettuare eventuali operazioni di ripulitura alla conclusione di una sessione di lavoro, un esempio di questo file è il seguente, sempre ripreso da una Debian Squeeze:

.bash_logout

```
# ~/.bash_logout: executed by bash(1) when login shell exits.

# when leaving the console clear the screen to increase privacy

if [ "$SHLVL" = 1 ]; then
    [ -x /usr/bin/clear_console ] && /usr/bin/clear_console -q
fi
```

in cui viene controllata la variabile `SHLVL` che indica il *livello* della shell per eseguire, se non si è in una *subshell*, le operazioni di ripulitura della console perché su questa non restino tracce della sessione precedente.

L'uso dei file `/etc/bash.bashrc` e `.bashrc` visto negli esempi precedenti, ci porta ad un'altra delle modalità di utilizzo della shell, quella in cui essa opera su linea di comando associata ad un terminale, ma senza essere un shell di login, come avviene ad esempio per le shell lanciate dagli emulatori di terminale messi in esecuzione all'interno dell'interfaccia grafica. In questo caso si ha quella che si chiama *shell interattiva*. La shell viene eseguita in modalità interattiva se invocata senza argomenti e con *standard output* e *standard input* associati ad un terminale, o se lanciata esplicitamente come tale con l'opzione `-i`. In tal caso viene automaticamente definita la variabile `PS1` e nella variabile speciale `$-` (vedi tab. 2.9) comparirà la lettera "`i`".

Se la shell è interattiva e non di login cambia semplicemente la modalità di inizializzazione; le impostazioni generali per tutti gli utenti vengono lette da `/etc/bash.bashrc` invece che da `/etc/profile`, mentre le impostazioni personali vengono lette direttamente da `.bashrc` invece che da uno dei tre file `.bash_profile`, `.bash_login` e `.profile`. Un esempio di `.bashrc`, ripreso dalla versione installata da Debian Squeeze per l'amministratore, è il seguente:

.bashrc

```
# ~/.bashrc: executed by bash(1) for non-login shells.

# Note: PS1 and umask are already set in /etc/profile. You should not
# need this unless you want different defaults for root.
# PS1='${debian_chroot:+($debian_chroot)}\h:\w\$ '
# umask 022

# You may uncomment the following lines if you want 'ls' to be colorized:
# export LS_OPTIONS='--color=auto'
# eval "'dircolors'"
# alias ls='ls $LS_OPTIONS'
# alias ll='ls $LS_OPTIONS -l'
```

```
# alias l='ls $LS_OPTIONS -lA'
#
# Some more alias to avoid making mistakes:
# alias rm='rm -i'
# alias cp='cp -i'
# alias mv='mv -i'
```

e come si può notare riguarda principalmente la possibilità, scommentando le relative linee, di definire degli alias ed un prompt specifico. Dato che una shell di login è anche interattiva, si capisce la precedente scelta per cui tutte le impostazioni comuni, sono demandate ai file `/etc/bash.bashrc` e `.bashrc` che vengono poi utilizzati direttamente da `/etc/profile` e da `.profile`.

Come si può notare le differenze fra queste diverse modalità di invocazione sono minime, e attinenti per lo più a come viene eseguita la procedura di inizializzazione di ambiente e variabili di shell. Tale procedura non è più necessaria (e pertanto non viene eseguita) tutte le volte che si riesegue la shell come *subshell* per lanciare uno script all'interno di una shell interattiva o di login.

Una terza modalità di funzionamento della `bash` si attiva quando questa viene invocata con il nome di `sh`, in tal caso infatti essa cercherà di simulare la caratteristiche di funzionamento di una shell POSIX elementare, per mantenere al massimo la compatibilità, e molte delle funzionalità avanzate non saranno disponibili. In questo caso ad esempio se la shell viene eseguita come shell di login, per le impostazioni saranno presi in considerazione solo `/etc/profile` e `.profile`.

Infine oltre alle modalità appena illustrate, `bash` supporta un'ultima modalità speciale, che si attiva invocandola con l'opzione `-r` o chiamandola direttamente come `rbash`: quella della cosiddetta *modalità ristretta*.[19] Invocando la shell in questa maniera è possibile creare un ambiente "*ristretto*" in cui essa è soggetta ad una serie di limitazioni che consentono di circoscrivere e controllare il suo funzionamento, un elenco delle limitazioni più importanti attive con questa modalità è il seguente:

- non sarà possibile cambiare directory con il comando `cd`.
- non si potranno modificare o cancellare le variabili di shell `SHELL`, `PATH`, `ENV`, o `BASH_ENV`.
- non si potranno specificare nomi di comandi contenenti il carattere "`/`".
- saranno bloccate le redirezioni dell'output.
- non si potrà usare il comando interno `exec` per eseguire un altro comando programma al posto della shell.

l'elenco completo, insieme ad una descrizione dettagliata di questa modalità di funzionamento (che può essere utile per la limitazione di eventuali account remoti) si può trovare nella sezione "`RESTRICTED SHELL`" della pagina di manuale.

Ma al di là di queste modalità generali, la `bash` consente di controllare alcuni aspetti più specifici del proprio funzionamento attraverso l'impostazione di alcune opzioni interne. Il comando principale che si usa per questo è `set`, che abbiamo già incontrato in sez. 2.1.3 per visualizzare le variabili e le funzioni di shell definite quando viene usato senza specificare nessuna opzione.

Il comando però consente di modificare vari aspetti del funzionamento della shell, modificando quelli che il manuale chiama gli "*attributi*" della shell. Ciascuno di questi è identificato da una

[19]benché nel manuale della `bash` si parli anche in questo caso di *restricted shell*, la cosa non è da confondere con l'omonimo concetto illustrato in sez. 4.3.5 che attiene ad una shell generica.

lettera e viene attivato specificando quest'ultima a modo di opzione preceduto dal carattere "`-`". Se invece si usa il carattere "`+`" al posto del "`-`" il relativo attributo verrà disattivato. Ad esempio abbiamo accennato in sez. 2.1.4 che se si esegue `set -C` la shell riporterà un errore, invece di eseguire la sovrascrittura, tutte le volte che si effettua una redirezione su un file esistente.

Un secondo attributo importante si abilita con `set -e` che attiva l'uscita immediata della shell qualora ci sia un errore in un comando. Si usa questa modalità principalmente negli script (vedi sez. 2.1.5) per interromperne automaticamente l'esecuzione in caso di errore, le uniche eccezioni sono quando gli errori avvengono in una istruzione condizionale, comprese quelle espresse con "`&&`" e "`||`".

Attributo	**Significato**
`-a`	inserisce automaticamente nell'ambiente qualunque variabile venga definita.
`-f`	se attivata disabilita il *filename globbing*.
`-e`	se attivata fa sì che la shell esca immediatamente in caso di errore nell'esecuzione di un comando.
`-u`	se attivata fa sì che la shell stampi un errore in caso di riferimento ad una variabile non definita.
`-C`	se attivata fa sì che la shell non sovrascriva un file esistente in caso di redirezione (vedi sez. 2.1.4) segnalando un errore.
`-o` *opt*	imposta l'opzione di funzionamento della shell passata come parametro.

Tabella 2.11: Principali attributi della shell, indicato come opzioni del comando interno `set`.

I principali attributi che si possono impostare con `set` sono riportati nella prima parte di tab. 2.11 in forma di opzione, per l'elenco completo e la descrizione dettagliata del significato di ciascuno di essi si può fare riferimento alla descrizione nella sezione "`SHELL BUILTIN COMMANDS`" della pagina di manuale.

Si è invece mantenuta a parte l'opzione `-o` perché costituisce una sorta di estensione degli attributi precedenti, e consente di attivare (o disattivare se usata come `+o`) le cosiddette "*opzioni*" della shell, che attengono ad un insieme molto più ampio di modalità operative. Queste devono essere indicate tramite un ulteriore parametro che indica il nome dell'opzione, si sono indicate le più rilevanti in tab. 2.12.

Come si può notare molte di queste opzioni ripropongono le impostazioni modificabili direttamente con gli attributi di tab. 2.11, con alcune aggiunte, come la possibilità di cambiare il comportamento delle combinazioni da tastiera per l'editor di linea per adattarsi a quelle usate da alcuni editor molto diffusi.

2.2 I comandi dei file

Dato che in un sistema unix-like tutto è un file, è naturale che la maggior parte dei comandi della "*unix toolbox*" abbia a che fare con le operazioni di manipolazione dei file. Abbiamo già visto in sez. 1.2 i comandi elementari che ci permettono la gestione dei file sul filesystem come `cp`, `mv`, `mkdir`, `ln`, `rm`, ecc. e quelli che consentono di operare sui metadati, come `chown`, `chmod`, ecc. In questa sezione affronteremo gli altri comandi che permettono di operare sul contenuto

Opzione	Significato
allexport	equivalente di set -a.
emacs	imposta l'editor di linea in stile emacs con combinazioni da tastiera riprese dall'editor emacs (il default di cui si è già parlato in sez. 2.1.3).
errexit	equivalente di set -e.
noclobber	equivalente di set -C.
noglob	equivalente di set -f.
nounset	equivalente di set -u.
posix	cambia il comportamento della shell in modalità di contabilità con lo standard POSIX.2.
vi	imposta l'editor di linea in stile vi, in cui le funzionalità di editing di linea usano combinazioni da tastiera riprese dall'editor vi (vedi sez. 2.3.3.

Tabella 2.12: Principali opzioni della shell, da indicare come parametro dell'opzione -o di set.

dei file ed eseguire su di essi operazioni più complesse; dato che buona parte di queste vengono effettuate con la redirezione, vedremo anche delle applicazioni di quest'ultima.

2.2.1 Caratteristiche comuni

Benché, come anticipato in sez. 2.1, ogni comando sia specializzato per fare un compito specifico, esistono comunque una serie di caratteristiche comuni a tutti loro. Dato che i comandi relativi ai file sono la maggioranza, le tratteremo qui, anche se si applicano in generale a qualunque tipo di comando, e non solo a quelli che riguardano i file.

La principale caratteristica comune a quasi tutti i comandi è la modalità di gestione delle opzioni. Abbiamo illustrato nei dettagli in sez. 2.1.3 come queste vengono elaborate dalla shell e passate come argomenti al comando e come la gran parte dei comandi le elabori attraverso una apposita libreria che ne assicura una sostanziale uniformità. Quello che c'è da aggiungere qui è che alcune di esse, le più comuni, tendono ad essere le stesse e ad avere lo stesso significato per tutti i comandi. Ad esempio per la gran parte dei comandi l'opzione -h può essere utilizzata per stampare a video una schermata di aiuto che riassume la sintassi di uso del comando.

Ricordiamo comunque che quella di identificare come opzioni gli argomenti che iniziano con il carattere "-" è solo una convenzione, e non è seguita da tutti i comandi, alcuni infatti usano delle sintassi diverse per eredità storiche, come abbiamo visto in sez. 1.3.2 con ps. La convenzione però viene usata estensivamente da tutti i comandi di base realizzati all'interno del progetto GNU (quelli del pacchetto coreutils) che hanno una certa uniformità anche nei nomi delle opzioni, e da quelli che usano le funzionalità di gestione fornite della libreria di sistema (la *GNU C library*); per questo è molto diffusa.

Come accennato in sez. 2.1.3 le opzioni sono di due tipi: con o senza parametri. Quando sono senza parametri esse si comportano come degli interruttori che attivano o disattivano una certa modalità di funzionamento; ad esempio l'opzione -v (che sta per *verbose*) viene usata da molti comandi per aumentare la *prolissità* dei propri messaggi. Le opzioni che prevedono dei parametri sono più complesse e agiscono in maniera analoga ad un cursore o una manopola; permettono cioè di passare dei valori di controllo per una certa funzionalità del comando. In tal

caso, con un formato del parametro che varierà da caso a caso, il valore dovrà essere specificato di seguito all'opzione, normalmente separato da uno spazio.

Inoltre, come accennato in sez. 2.1.3, i comandi del progetto GNU supportano anche una versione estesa delle opzioni, in cui queste si possono specificare con parole intere invece che con singole lettere. In questo caso esse iniziano sempre con un "--". Tutti i comandi di base del progetto GNU ad esempio supportano le due opzioni `--help`, che stampa una schermata riassuntiva della sintassi (equivalente a `-h`), e `--version`, che stampa il numero di versione. Se l'opzione estesa deve indicare un valore questo viene normalmente specificato in forma di assegnazione con un "=", ad esempio con qualcosa del tipo `--tabsize=80`, anche se spesso basta fornire il valore separato da uno spazio.

Si ricordi che l'interpretazione delle opzioni è un compito strettamente attinente al singolo programma, per la shell un'opzione è una stringa come un'altra che viene passata al comando che viene eseguito. Nasce allora il problema di quando il valore di un certo argomento, come ad esempio il nome di un file, inizia con il carattere "-", e non si ha modo di distinguerlo da una opzione. Dato che sulla linea di comando si possono passare come argomenti anche i risultati delle varie espansioni (di comandi, variabili o nomi di file), talvolta questo tipo di inconvenienti può verificarsi in maniera inaspettata.

Per questo motivo la combinazione "--" usata da sola viene utilizzata dalle librerie di gestione delle opzioni per forzare la conclusione della scansione delle opzioni nella riga di comando, in modo che tutti gli argomenti che seguono ad essa vengano usati dal programma letteralmente, senza considerarli possibili opzioni, anche se cominciano per un "-". Si tenga presente però che questa è un'altra delle funzionalità introdotte dalla funzione `getopt` della *GNU C Library*, usata dalla gran parte dei programmi, ma non da tutti, per gestire le opzioni a riga di comando.

Come esempio d'uso di questa funzionalità si consideri di dover cancellare con `rm` un file il cui nome inizia per "-" (ad esempio `-lista`). Dato che il "-" con cui inizia il nome normalmente viene interpretato come l'inizio di una opzione, l'uso di `rm -lista` produce un errore essendo l'opzione `-l` inesistente. Proteggere il "-" con la barra, o usare le virgolette per il nome non serve a nulla, la shell interpreterà correttamente quanto indicato, passando comunque il valore `-lista` come argomento.

Il problema infatti non deriva dalla shell, ma da come vengono gestite le opzioni nei comandi: tutti i programmi infatti prima esaminano le opzioni, segnalando errori se ne trovano di inesistenti, e poi trattano gli argomenti. Per questo occorre indicare esplicitamente al comando che in questo caso non si vuole che il "-" sia interpretato come l'inizio di una opzione: scrivendo il comando come `rm -- -lista`, l'interpretazione delle opzioni viene bloccata dal "--" e a questo punto `-lista` (anche se inizia per "-") verrà considerato un normale argomento che indica il nome di un file, che sarà cancellato.

Un'ultima convenzione, seguita anch'essa da molti programmi, riguarda l'uso del singolo carattere "-" come argomento. La convenzione viene utilizzata dai comandi che prendono come argomento un nome di file da cui leggere o su cui scrivere dati. Se al posto del nome di un file si usa questo carattere il comando farà riferimento rispettivamente allo *standard input* per la lettura o allo *standard output* per la scrittura.

2.2.2 I comandi per le ricerche dei file

La ricerca di uno specifico file all'interno del filesystem, o dei file con una certa serie di caratteristiche, è una operazione molto comune, e per questo sono stati sviluppati alcuni comandi molto flessibili, che permettono di effettuare le più complesse tipologie di ricerca.

Un primo comando che si può usare per cercare dei file l'abbiamo già incontrato in sez. 2.1.3, ed è `which`, che ci indica a quale file eseguibile corrisponde un certo comando, facendo una ricerca nelle directory indicate nel *PATH*. Una estensione di `which`, è il comando `whereis` che consente di estendere la ricerca e ricercare oltre ai binari eseguibili anche le pagine di manuale ed eventuali sorgenti associati ad un comando.

Il comando `whereis` prende come argomento il nome del file o comando da cercare, e stampa i risultati sullo *standard output*. Si possono restringere le ricerche soltanto a binari, pagine di manuale o sorgenti utilizzando rispettivamente le opzioni `-b`, `-m` e `-s`. Il comando prende il nome passato come argomento, elimina eventuali estensioni (ad esempio se si cerca un sorgente elimina il `.c`) ed esegue la ricerca in una serie di directory standard (elencate nella pagina di manuale). Si possono restringere le ricerche a delle directory specifiche per binari, pagine di manuale e sorgenti usando rispettivamente le opzioni `-B`, `-M` e `-S` seguite dalle directory a cui restringere la ricerca. Per tutti i dettagli e l'elenco completo delle opzioni si consulti al solito la pagina di manuale.

Il comando più veloce per cercare un file in maniera generica sul filesystem è `locate`, che, come suggerisce il nome, serve a localizzare sul filesystem tutti i file che contengono nel loro pathname la stringa passata come argomento. Il vantaggio di questo programma è la sua velocità, esso infatti non effettua la ricerca con una scansione del contenuto del disco, ma su un piccolo database interno che contiene l'elenco di tutti i file presenti nel sistema. Di questo programma esistono diverse versioni che si sono succedute nel tempo, qui prenderemo in considerazione quella che viene fornita dal pacchetto `mlocate`, adottata da tutte le principali distribuzioni.

Il comando riconosce l'opzione `-i`, che richiede che venga effettuata una ricerca *case insensitive*, e `-e` che richiede che sia verificata l'effettiva esistenza del file. Il comando inoltre consente l'uso come argomento di espressioni analoghe a quelle usate dalla shell per il *filename globbing* viste in sez. 2.1.3, se si usa l'opzione `-r` si potrà usare, come unico argomento, una espressione regolare (vedi sez. 2.2.5). Le altre opzioni e la descrizione completa del comando è al solito disponibile nella relativa pagina di manuale, accessibile con `man locate`.

Il fatto che `locate` si affidi ad un database ci fa capire immediatamente anche i suoi limiti: anzitutto la ricerca può essere effettuata solo per nome, ed inoltre il comando sarà in grado di effettuare la ricerca solo sui file già inseriti nel database. Questo viene creato ed aggiornato tramite il comando `updatedb` che in genere viene eseguito una volta al giorno (fra i lavori periodici di `cron` che vedremo in sez. 3.2.1), per cui se un file è stato creato dopo l'ultimo aggiornamento non si potrà vederlo.

Il comando `updatedb` prende una serie di opzioni in forma estesa, le principali delle quali sono illustrate in tab. 2.13, che gli permettono di definire quale file usare per il database,[20] quali directory (ad esempio non ha molto senso indicizzare `/tmp`) e quali filesystem (non ha senso indicizzare un CDROM o un filesystem di rete) inserire nel database dei file e quali no.

La caratteristica di `updatedb` è che i valori di alcune opzioni (quelle riportate nella seconda parte di tab. 2.13) possono essere impostate attraverso l'uso del file di configurazione,

[20] in genere (ad esempio per Debian e RedHat) viene usato `/var/lib/mlocate/mlocate.db`.

Opzione	Significato
--output	indica il file su cui viene salvato il database.
--prunefs	indica l'elenco dei tipi di filesystem da non indicizzare (il parametro deve essere passato come lista separata da spazi dei rispettivi nomi, come riportati in tab. 5.2).
--prunepaths	indica la lista delle directory da non indicizzare (il parametro deve essere passato come lista separata da spazi dei rispettivi pathname assoluti).
--prunenames	indica la lista di nomi di directory da non indicizzare (il parametro deve essere passato come lista separata da spazi) non sono consentiti pathname assoluti né caratteri jolly.

Tabella 2.13: Principali opzioni di `updatedb`.

`/etc/updatedb.conf`. Il file prevede che i valori siano assegnati in forma di assegnazione di un valore ad una variabile di shell, con i nomi delle variabili identici a quelli delle opzioni corrispondenti, ma scritte in maiuscolo anziché minuscolo e senza il `--` iniziale. Un contenuto tipico di questo file (così come ripreso dalla versione installata su una Debian Squeeze) è il seguente:

/etc/updatedb.conf

```
PRUNE_BIND_MOUNTS="yes"
# PRUNENAMES=".git .bzr .hg .svn"
PRUNEPATHS="/tmp /var/spool /media"
PRUNEFS="NFS nfs nfs4 rpc_pipefs afs binfmt_misc proc smbfs autofs iso9660
 ncpfs coda devpts ftpfs devfs mfs shfs sysfs cifs lustre_lite tmpfs usbfs
 udf fuse.glusterfs fuse.sshfs"
```

Allora, in corrispondenza con le omonime opzioni di tab. 2.13, con `PRUNEFS` si indicano i filesystem da non indicizzare, come quelli di rete, i filesystem virtuali come `/proc` o quelli di dispositivi estraibili mentre con `PRUNEPATHS` si indicano le directory da non indicizzare, come quelle dei file temporanei, gli *spool* delle code di stampa e della posta, ecc. L'opzione `PRUNE_BIND_MOUNTS` è invece un valore logico, che consente di eliminare le apparizioni duplicate degli stessi file che si trovano al di sotto dei eventuali *bind mount* (vedi sez. 5.1.3).

Un problema che si poneva con la prima versione di `locate` era che il comando restituiva tutte le voci indicizzate indipendentemente dai loro permessi, permettendo così ad un utente normale di verificare la presenza di file per i quali non avrebbe accesso. Questo veniva risolto a livello di indicizzazione eseguendo quest'ultima per conto di un utente senza privilegi sul filesystem in modo da inserire nel database solo i file visibili pubblicamente, con la conseguenza però che gli utenti non erano in grado di avere indicizzati i propri file privati. Per risolvere questo problema le versioni attuali del programma eseguono una indicizzazione generale e mostrano i risultati verificando se i permessi dell'utente consentono l'accesso al file.

Per superare i limiti di `locate` e affini, si può usare il comando `find`, che non utilizza un database, ma esegue la ricerca direttamente nel filesystem, al costo di una notevole attività su disco, e di tempi di esecuzione decisamente più lunghi. Si può comunque ridurre il carico facendo effettuare la ricerca su sezioni ridotte dell'albero dei file; il comando infatti prende come primo argomento la directory da cui iniziare la ricerca, che verrà eseguita ricorsivamente in tutte le directory sottostanti; se non si specifica nulla la ricerca partirà dalla directory corrente.

Il comando supporta quattro categorie principali di opzioni, descritte da altrettante sezioni della pagina di manuale, accessibile al solito con `man find`. La prima categoria (descritta nella sezione `OPTIONS`) contiene le opzioni vere e proprie, che controllano il comportamento di `find`, la seconda, (descritta nella sezione `TESTS`) contiene le opzioni di ricerca che permettono di selezionare i file in base ad una loro qualunque proprietà (nome, tipo, proprietario, i vari tempi, permessi, ecc.), la terza (descritta nella sezione `ACTIONS`) contiene le opzioni che permettono di specificare una azione da eseguire per ciascun file che corrisponde alla ricerca, la quarta (descritta nella sezione `OPERATORS`) contiene le opzioni che permettono di combinare fra loro diverse selezioni.

Opzione	**Significato**
`-L`	dereferenzia i link simbolici, sostituisce la precedente `-follow`, deprecata.
`-mount`	resta nel filesystem corrente e non analizza sottodirectory in altri filesystem.
`-maxdepth` *n*	limita la profondità della ricerca ad *n* directory sotto quella di partenza.
`-mindepth` *n*	imposta la profondità iniziale della ricerca a *n* directory sotto quella di partenza.

Tabella 2.14: Principali opzioni generiche di `find`.

Le opzioni generiche, le principali delle quali sono riportate in tab. 2.14, permettono di modificare il comportamento del comando, ad esempio con `-maxdepth` si può limitare la ricerca ai primi livelli di sottodirectory, mentre con `-mindepth` la si può far partire da un certo sottolivello. Inoltre con `-L` si può richiedere che il comando risolva eventuali link simbolici, facendo riferimento ai file cui essi puntano anziché al link stesso, come avviene di default.

Le maggiori potenzialità di `find` derivano dalla sua capacità di effettuare ricerche con i criteri più svariati, da quelli sul nome del file in varie forme (con `-name`, `-regex`, `-path`), a quelli per gruppo e utente (con `-group` e `-user`), secondo i permessi (con `-perm`), secondo i vari tempi (`-atime`, `-ctime`, `-mtime`) per tipo di file, di filesystem su cui è il file, ecc. Un elenco delle principali opzioni di ricerca, con il relativo significato, è riportato in tab. 2.15.

Alcune di queste opzioni necessitano di alcuni chiarimenti, ad esempio con l'opzione `-name` si può effettuare la classica ricerca sul nome del file, con tanto di supporto per i caratteri jolly del *filename globbing*, che però vanno adeguatamente protetti scrivendoli fra virgolette per evitarne l'espansione da parte della shell. La ricerca è effettuata solamente sul nome del file così come è scritto nella directory che lo contiene, e non sul suo pathname, se si vuole eseguire la ricerca su quest'ultimo occorre usare `-path`.

Per tutte le opzioni che prendono un valore numerico che si è indicato in tab. 2.15 con "`n`", come quelle sui tempi, gli identificatori, il numero di link, ecc. il comando permette una sintassi molto potente: specificando solo il numero si richiede una corrispondenza esatta, precedendolo con il segno "`-`" si richiede invece che il valore sia inferiore, mentre precedendolo con un "`+`" si richiede che sia superiore. In questo modo ad esempio, nel caso dei tempi, si può richiedere che un file sia più vecchio o più giovane di un dato tempo. Per cui se si vogliono cercare i file modificati negli ultimi 5 minuti si dovrà dare il comando:

```
piccardi@anarres:~/truedoc/corso$ find . -mmin -5
.
```

Opzione	Significato
`-amin` *n*	un file acceduto *n* minuti fa, le opzioni `-cmin` e `-mmin` eseguono lo stesso controllo rispettivamente coni tempi di ultimo cambiamento e ultima modifica.
`-atime` *n*	un file acceduto *n* giorni fa, le opzioni `-ctime` e `-mtime` eseguono lo stesso controllo rispettivamente con i tempi di ultimo cambiamento e ultima modifica.
`-anewer` *file*	un file acceduto più recentemente di *file*, le opzioni `-cnewer` e `-mnewer` eseguono lo stesso controllo rispettivamente con i tempi di ultimo cambiamento e ultima modifica.
`-gid` *n*	il *group ID* del gruppo proprietario è *n*.
`-group` *group*	il gruppo proprietario è *group*.
`-links` *n*	il file ha *n* *hard link*.
`-name` *pattern*	il nome del file corrisponde al pattern *pattern*. Prevede anche `-iname` per una ricerca case insensitive.
`-path` *pattern*	il pathname del file (comprese quindi le directory a partire dalla radice) corrisponde al pattern *pattern*. Prevede anche `-ipath` per una ricerca case insensitive.
`-perm` *mode*	i permessi corrispondono a *mode*.
`-size` *n*	la dimensione del file è *n*.
`-type` *c*	seleziona sul tipo di file, il valore di *c* corrisponde alla lettera usata da `ls` e riportata in tab. 1.1.
`-uid` *n*	l'*user ID* del proprietario è *n*.
`-user` *user*	il proprietario è *user*.

Tabella 2.15: Principali opzioni di `find` per la ricerca.

```
./shell.tex
```

mentre se si vuol cercare quelli non acceduti da più di quindici giorni si farà:

```
piccardi@anarres:~/truedoc/corso$ find . -atime +15
./ringraziamenti.tex
```

L'opzione di ricerca sui permessi `-perm` merita una trattazione a parte, il parametro `mode` passato all'opzione supporta infatti a sua volta una sintassi abbastanza complessa. Anzitutto un permesso può essere espresso sia in forma simbolica, con lo stesso formato visto in sez. 1.4.3, ma usando solo "`=`" come operatore, che in forma numerica. Specificando un valore `find` cercherà un file che abbia esattamente quel valore per i permessi, ad esempio se si specifica `g=w` corrisponderanno soltanto i file con un permesso numerico esattamente uguale a `020`.

Dato che in genere non si è interessati alla ricerca di un valore specifico di tutti i permessi, quanto alla presenza di alcuni di essi, indipendentemente da quali possono essere gli altri, il valore può essere specificato apponendovi, in maniera simile a quanto si fa con i valori numerici, i due segni "`-`" e "`+`", in realtà quest'ultimo è ormai deprecato, ed al suo posto occorre usare "`/`". Trattandosi però nel caso dei permessi di una maschera di bit, il significato di questi caratteri non può essere come in precedenza relativo ad un valore numerico maggiore o minore di quello indicato, considerato poi che la notazione è valida anche per le espressioni simboliche.

Dato che un valore espresso senza segno richiede la corrispondenza esatta, si usano queste combinazioni per selezionare la presenza di uno o più bit senza curarsi dello stato degli altri, che è in genere il tipo di ricerca più utile. Se si usa il segno "`-`" allora `mode` dovrà specificare, se si usa la forma simbolica, quali sono i permessi che devono essere presenti sul file, oppure se

si usa la maschera dei bit quali si vuole che siano presenti (secondo lo schema di fig. 1.6), i bit nulli e gli altri permessi verranno ignorati. Se invece si usa "`/`" la richiesta è più debole ed il file corrisponde purché almeno uno dei permessi specificati con il valore di `mode` sia attivo. In questo modo con `-mode` si può richiedere una condizione in cui siano attivi un permesso "*e*" un altro, mentre con `/mode` una in cui siano attivi un permesso "*o*" un altro.

Come accennato una seconda importante categoria di opzioni è quella relativa alle azioni; è possibile infatti, per ogni file che corrisponde al criterio di ricerca specificato, far eseguire a `find` una certa azione. Se non si specifica nulla l'azione di default è quella di stampare il nome del file, equivalente alla opzione `-print`; ma si possono anche scrivere i nomi su un file qualunque usando l'opzione `-fprint file`, o usare vari formati di stampa.

Opzione	**Significato**
`-exec`	esegue un comando usando come argomento il nome del file.
`-print`	stampa il nome del file terminato con un a capo.
`-print0`	stampa il nome del file terminato con un carattere NUL (il valore 0).
`-fprint` *`file`*	scrive il nome del file sul file *`file`*.
`-ok`	come `-exec` ma chiede conferma del comando.

Tabella 2.16: Principali opzioni di `find` per specificare azioni.

L'elenco delle principali opzioni per le azioni è riportato in tab. 2.16. Quella di gran lunga più importante è `-exec` che permette di eseguire, per ogni file corrispondente alla selezione, un comando. La sintassi dell'opzione è complessa in quanto si deve inserire una riga di comando all'interno di un'altra, e ci sono delle convenzioni usate dal comando per passare i valori. Quando si usa `-exec` tutto quello che segue viene interpretato come una riga di comando fino a che non si incontra un carattere "`;`", in detta riga si può fare riferimento al file che corrisponde con la stringa `{}`. Il problema è che tutti questi caratteri vengono interpretati dalla shell, e devono quindi essere adeguatamente protetti; allora se ad esempio si vogliono spostare tutti i file non acceduti da più di 15 giorni in una directory `old`, si potrà usare un comando del tipo:

```
find . -atime +15 -exec mv \{\} old \;
```

La potenza del comando `find` è poi ulteriormente aumentata dal fatto che le varie opzioni precedenti possono essere combinate fra di loro con degli operatori logici. Ma se il significato di `-and` o `-or` può sembrare immediato nel caso di criteri di ricerca, diventa meno chiaro quando si ha a che fare con delle azioni. In realtà infatti il comando associa un valore logico ad ogni opzione, e quando si esegue una selezione il valore è automaticamente vero, lo stesso vale per tutte le azioni, tranne `-exec` (e derivate come `-ok`) in cui il valore è vero se il comando ha uno stato di uscita nullo, e falso altrimenti.

Il funzionamento di un operatore come `-and` (che è sottinteso se si specificano più opzioni) è che la seconda opzione (sia questa di ricerca, che una azione) viene eseguita solo se la prima è vera. Viceversa con `-or` la seconda opzione viene eseguita solo se la prima è falsa. Infine `-not` nega il risultato di una opzione.

Nel caso si combinino opzioni di ricerca tutto questo è del tutto ininfluente riguardo il risultato del comando, che è quello che ci si aspetta intuitivamente: entrambe le condizioni di ricerca devono essere soddisfatte per `-and` o solo una per `-or`, o si inverte la selezione con `-not`. Le cose

cambiano profondamente quando ci sono di mezzo delle azioni come `-exec`, perché in tal caso l'esecuzione della seconda opzione dipende dal risultato della prima: se si chiede di eseguire due comandi ad esempio le cose dipendono dal risultato di quello che si esegue per primo.

Per questo ad esempio specificare con `-and` più comandi (o semplicemente scriverne più di uno, dato che in tal caso il `-and` è sottinteso) non significa affatto che essi saranno eseguiti tutti: lo saranno solo se tutti hanno successo, se uno non ha successo i successivi non saranno eseguiti. Qualora si voglia essere sicuri di eseguire tutti i comandi in una lista si può usare l'operatore "`,`" nel qual caso saranno eseguiti comunque tutti, ma si avrà un valore finale corrispondente all'ultimo della lista.

2.2.3 I comandi per visualizzare il contenuto dei file

Si sono raccolti in questa sezione tutti i comandi che permettono di visualizzare il contenuto di un file. Il primo comando di questo tipo è `cat`, che abbiamo già incontrato in sez. 2.1.4. In tale occasione abbiamo anche accennato come in realtà l'uso principale di questo comando (a parte il caso di file molto corti) sia quello di permettere la concatenazione del contenuto di più file, e che se lo scopo è solo quello della visualizzazione del contenuto di un file esistono alternative migliori; adesso tratteremo proprio queste.

Il problema maggiore dell'uso di `cat` come visualizzatore è che questo scrive tutto sul terminale, senza possibilità di mostrare il contenuto del file un po' alla volta. Per questo sono stati allora creati tutta una serie di programmi studiati per mostrare il contenuto dei file una *pagina* alla volta (dove per pagina si intende la schermata del terminale), che per questo sono detti *pager*. Ad essi è dedicata anche una variabile di ambiente, `PAGER`, usata dai programmi che necessitano di visualizzare il contenuto di un file, per scegliere quale fra i vari comandi di visualizzazione deve essere utilizzato.

Il primo programma realizzato per la visualizzazione di interi file è `more`, il quale prende come argomento una lista di file da leggere di cui stampa il contenuto sul terminale una pagina alla volta, attendendo che l'utente invii dei comandi da tastiera. Al solito la pagina di manuale riporta l'elenco completo delle opzioni ed i vari comandi usati per controllare il comportamento del programma. Ad esempio con `-num` si può specificare un parametro che indica il numero di linee che devono essere stampate sullo schermo (utile solo quando il comando non riesce a determinarlo da solo). Rimandiamo ad essa per le informazioni complete, qui faremo solo una breve panoramica sui principali comandi che si possono dare durante la visualizzazione, il cui elenco comunque può essere ottenuto direttamente durante l'uso del programma premendo i tasti "`?`" o "`h`".

Una volta stampata una pagina `more` consente di passare a quella successiva con la pressione dello spazio, mentre l'uso del ritorno a capo permette di avanzare lo scorrimento di una riga alla volta. Si può interrompere la visualizzazione con "`q`", mentre con "`b`" si può tornare una pagina indietro. Se si sono indicati più file con `:n` si può passare alla visualizzazione del successivo mentre con `:p` tornare al precedente. Con il tasto "`/`" si fa apparire un prompt dove inserire una stringa da ricercare all'interno del file. In realtà la stringa viene interpretata come una *espressione regolare* (vedi sez. 2.2.5), quindi si possono effettuare anche ricerche molto complesse. Infine con "`v`" si può lanciare l'editor impostato con la variabile di ambiente `EDITOR` per modificare il contenuto del file (gli editor sono trattati in sez. 2.3, quello usato di default è `vi`).

Il comando more è stato creato fin dagli albori di Unix, e la sua sintassi risente anche del fatto che i primi terminali erano delle telescriventi, su cui il testo veniva stampato, per cui non supporta ad esempio l'uso dei tasti di freccia avanti e indietro ed i vari tasti di spostamento presenti nelle tastiere moderne. Dato che ben presto tutti i terminali iniziarono a supportare la riscrittura dello schermo, e tutte le tastiere iniziarono ad avere i tasti di spostamento, venne creato less come *evoluzione*[21] di more.

Le funzionalità di less sono sostanzialmente identiche a quelle di more, e supporta anche tutti i comandi da tastiera illustrati in precedenza, ma il programma consente degli spostamenti più comodi, potendo navigare il contenuto del file avanti ed indietro con i tasti di freccia, pagina su e giù, ecc. Il comando poi supporta funzionalità avanzate come la possibilità di ridefinire dei *keybinding*, di lanciare dei programmi per pre-processare dei dati (ad esempio decomprimere al volo dei file compressi), ecc. Per i dettagli si faccia al solito riferimento alla pagina di manuale.

Un altro programma di visualizzazione, più utile in caso di file binari, è od, (da ***o**ctal **d**ump*) che permette di stampare il contenuto di un file in forma numerica, usando vari formati: decimale, ottale (il default), esadecimale e pure direttamente in semplice ASCII. La modalità in cui viene stampato il contenuto è controllata dall'opzione -t, che prende come parametro una stringa indicante il formato, il cui primo carattere indica il tipo di rappresentazione scelta, secondo quanto riportato in tab. 2.17. Nel caso si sia usata una forma numerica, si può utilizzare un secondo carattere per indicare la dimensione in byte del numero in questione.

Carattere	**Formato**
a	caratteri ASCII, coi caratteri non stampabili riportati tramite un nome simbolico.
c	carattere ASCII, coi caratteri non stampabili riportati in forma numerica preceduta dalla barra rovescia.
d	decimale.
f	virgola mobile.
o	ottale.
u	decimale senza segno.
x	esadecimale.

Tabella 2.17: I caratteri indicanti il formato per la stampa dell'output del comando od.

Una seconda opzione che permette di modificare il formato di stampa di od è -A, che stabilisce come deve essere stampato il numero progressivo che indica la posizione nel file; l'opzione prende come parametro uno dei caratteri d, o, x, o n dove i primi tre hanno lo stesso significato riportato in tab. 2.17 mentre n indica che non deve essere stampato nulla.

Altre due opzioni utili sono -j che permette di iniziare la stampa a partire da una certa posizione all'interno del file e prende come parametro il numero di byte da saltare, e -N che permette di specificare, passandolo come parametro, il numero di byte da stampare, altrimenti il comando stampa tutto il contenuto del file. Per le altre opzioni ed i dettagli di funzionamento del comando si faccia al solito riferimento alla pagina di manuale.

Del tutto analogo ad od per funzionalità e utilizzo è il comando hexdump che esegue le stesse operazioni ma che, nascendo come ***hex**adecimal **dump***, di default stampa il risultato in formato esadecimale anziché ottale. Come il precedente il comando, che può essere invocato anche come

[21] si, volevano davvero fare gli spiritosi!

hd, prende una serie di opzioni che consentono di controllare la visualizzazione, riportate nella seconda parte di tab. 2.18.

Opzione	Significato
-n *N*	stampa solo il numero di caratteri indicato come parametro (il default è stampare fino alla fine del file).
-s *N*	inizia a stampare dal numero di caratteri indicato come parametro rispetto all'inizio del file.
-e *str*	indica una stringa di formattazione per l'uscita del comando (per il formato si consulti la pagina di manuale).
-v	stampa il contenuto completo (in caso di righe consecutive identiche di default queste vengono sostituite con una riga di asterischi).
-c	stampa il contenuto in caratteri ASCII, coi caratteri non stampabili riportati in forma numerica (o simbolica) preceduti dalla barra rovescia.
-d	stampa il contenuto in forma decimale intera (usando due byte alla volta) con 8 valori per riga.
-b	stampa il contenuto in forma ottale (con ciascun byte indicato da tre cifre separate da spazi) con 16 valori per riga.
-x	stampa il contenuto in forma esadecimale (usando due byte alla volta) con 8 valori per riga.
-C	stampa il contenuto con una combinazione di esadecimale e ASCII.

Tabella 2.18: Le principali opzioni di hexdump.

Oltre a quelle di visualizzazione altre opzioni significative sono -s e -n che consentono di indicare rispettivamente posizione di partenza e numero di byte da stampare. Inoltre rispetto ad od il comando consente di controllare il formato di stampa in maniera molto dettagliata, in particolare l'opzione -e che consente di indicare una stringa di formattazione. Per l'elenco completo delle opzioni ed i dettagli di funzionamento, ed in particolare per il formato delle stringhe di formattazione, si faccia riferimento alla pagina di manuale.

Un'altra particolare forma di visualizzazione del contenuto di un file viene eseguita dal comando tac, che serve a stampare il contenuto di un file alla rovescia, cioè a partire dall'ultima riga verso la prima.[22] Il comando in realtà divide un file in campi separati da un carattere, che di default è il ritorno a capo per cui i campi vengono a coincidere con le righe; con l'opzione -s però si può passare come parametro un diverso separatore, mentre con -r si può usare come separatore una espressione regolare (vedi sez. 2.2.5). Per i dettagli conviene consultare la pagina *info* con info tac dato che la pagina di manuale è molto sintetica.

Un'ultima forma particolare di visualizzazione del contenuto di un file è quella ottenibile con il comando wc, (da ***word count***) che viene usato per contare linee, parole e caratteri contenuti in un file. Il comando prende come argomento una lista di file (se non se ne specificano al solito viene usato lo *standard input*) per ciascuno dei quali stampa il numero totale di linee, di parole e byte. In genere il comando stampa tutte queste informazioni insieme al nome del file, se ne sono specificati più di uno. Si può far stampare solo il numero di linee, di parole o di byte rispettivamente con le opzioni -l, -w e -c. L'opzione -L invece stampa la lunghezza della linea più lunga. Al solito per tutti i dettagli sul comando si consulti la pagina di manuale.

[22] si, l'idea era proprio quella di fare cat al contrario...

2.2.4 I comandi di elaborazione dei contenuti dei file

In questa sezione prenderemo in esame una serie di comandi che permettono di filtrare e manipolare in maniera automatica il contenuto dei file. Questi programmi costituiscono uno strumento estremamente potente e sono il cuore della cosiddetta *Unix toolbox*, in quanto il loro inserimento all'interno di una concatenazione di comandi permette di eseguire delle operazioni di elaborazione dei contenuti di un flusso di dati in maniera rapida ed efficiente, costruendo, grazie alle funzionalità del *pipelining* della shell, quella *catena di montaggio* di cui parlavamo in sez. 2.1.1.

Una prima serie di possibili comandi di elaborazione sono quelli che riguardano la suddivisione, in varie modalità, dei contenuti di un file. I primi due comandi che prenderemo in esame sono `head` e `tail`, che ci permettono di selezionare e poi scrivere sullo *standard output* rispettivamente l'inizio e la fine di un file (o di una lista di file). Entrambi i programmi usano l'opzione `-n` per indicare il numero di linee totali da selezionare (il default è 10), e l'opzione `-c` per effettuare la selezione in byte invece che in linee. Entrambi i comandi, quando non si specifica nessun argomento, operano sullo *standard input*. Al solito si faccia riferimento alla pagina di manuale per l'elenco completo delle opzioni e la descrizione dettagliata dei comandi.

Abbiamo visto come ottenere l'inizio e la fine di un file, ma la nostra cassetta degli attrezzi sembrerebbe mancare di un comando che ci permetta di selezionare una sezione qualunque del file a partire da una certa riga `N` per finire con un'altra `M`. In realtà questo ancora una volta è facilmente ottenibile concatenando i due comandi precedenti; basterà tagliare prima la testa del file con `head` e poi la coda con `tail`, costruendo con l'ausilio della *arithmetic expansion* che abbiamo descritto in sez. 2.1.3 una linea di comando del tipo:

```
head -n M file | tail -n $((M-N+1))
```

Vale la pena poi menzionare esplicitamente l'opzione `-f` di `tail`, che quando usata fa sì che il comando non esca, e continui a stampare ogni eventuale altro dato aggiunto in coda al file, permettendo così di seguire la crescita di quest'ultimo. Questa è una opzione molto utile per tenere sotto controllo i file di log (vedi sez. 3.2.3), ed in generale tutti i file in cui altri programmi scrivono in coda i loro dati.

In sez. 2.1.4 abbiamo visto come usare `cat` per concatenare il contenuto di più file, per eseguire l'operazione inversa si può usare `split`, che viene usato per "*tagliare a fette*" un file. Il comando prende come argomento il file da "*affettare*", e lo suddivide in tanti file di dimensione uguale che chiama progressivamente `xaa`, `xab`, `xac`, ecc. Se non si specifica un argomento al solito `split` legge dallo *standard input*, consentendo così ad esempio di creare un file con un comando e suddividerlo al volo con una *pipe*. Dato che il comando crea le sue *fette* in ordine alfabetico, per ricomporre il file originario basterà usare `cat` con un comando del tipo `cat x* > file`.

Aggiungendo un secondo argomento dopo il nome del file da suddividere si può specificare un prefisso diverso da `x` come intestazione dei nuovi file creati dal comando. La dimensione dei file viene specificata con l'opzione `-l` se la si vuole in numero di linee o con `-b` se la si vuole in numero di byte; quest'ultima opzione supporta anche un valore del parametro con i suffissi `m` e `k` per indicare rispettivamente megabyte e kilobyte. Infine se due lettere non bastano per indicizzare i file che si generano si può usare l'opzione `-a` per specificarne un numero diverso. Un elenco delle principali opzioni è riassunto in tab. 2.19, per tutti i dettagli si faccia al solito riferimento alla pagina di manuale.

Opzione	Significato
`-b N`	specifica la dimensione delle "*fette*" in byte.
`-a N`	specifica la lunghezza del suffisso.
`-d`	usa un suffisso numerico anziché alfabetico.
`-l N`	specifica la dimensione delle "*fette*" in linee.

Tabella 2.19: Le principali opzioni del comando `split`.

Se si vuole *affettare* un file per colonne (cioè in *verticale*) invece che per righe (in *orizzontale*) si può usare il comando `cut`. Il comando opera sul file passato come argomento, o sullo *standard input*, rendendo di nuovo possibile operazioni complesse e filtri ricorsivi; le colonne selezionate verranno stampate sullo *standard output*.

Con l'opzione `-c` si può creare la selezione scegliendo quali caratteri inserire nel risultato in base alla loro posizione rispetto all'inizio della riga. L'opzione prende una lista di posizioni numeriche separata da virgole, e supporta la presenza di intervalli indicati con due numeri separati da un "`-`" (il comando conta da 1 per il carattere ad inizio riga). Inoltre se si specifica un valore preceduto da "`-`" si selezionerà tutta la riga dall'inizio a quella posizione, e viceversa un valore seguito da "`-`" selezionerà tutto da quella posizione a fine riga. Così se ad esempio si vuole ottenere la stringa dei permessi che sta all'inizio dall'output di `ls -l` basterà eseguire:

```
piccardi@anarres:~/truedoc/corso$ ls -l *.tex | cut -c 1-10
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
```

L'utilità del comando è che oltre ad una selezione basata su posizioni assolute all'interno della riga, esso permette di effettuare la selezione anche in termini di *campi* delimitati da un carattere qualunque che può essere specificato con l'opzione `-d` (di default il comando usa come separatore il carattere di tabulazione). In tal caso si effettuerà la selezione di quali campi stampare con l'opzione `-f`, indicando i numeri dei campi scelti con la stessa sintassi usata per `-c`, cioè con liste separate da virgole ed intervalli. Così si potrà ad esempio stampare il proprio *user ID* con:[23]

```
piccardi@anarres:~$ cat /etc/passwd | grep piccardi | cut -d: -f 3
1000
```

Si tenga presente che le selezioni per campi o caratteri sono alternative fra loro. Le opzioni principali del comando sono riportate in tab. 2.20, al solito per i dettagli di funzionamento e l'elenco completo si consulti la pagina di manuale.

Oltre a suddividere o selezionare parte del contenuto di un file, esistono molte situazioni in cui occorre eseguire manipolazioni di altra natura, ed un secondo insieme di comandi è stato fornito a questo scopo. Come contraltare a `cut`, il comando `paste` permette di concatenare file

[23] il formato del file `/etc/passwd` è trattato in sez. 4.3.2, e vedremo `grep`, che consente di effettuare la ricerca di una stringa in un file, in sez. 2.2.5.

Opzione	Significato
`-c N`	specifica un intervallo di caratteri.
`-d c`	specifica il carattere delimitatore dei campi.
`-f N`	specifica un intervallo di campi.
`-b N`	specifica un intervallo di byte (sull'intero file).

Tabella 2.20: Le principali opzioni del comando `cut`.

diversi in colonna. Il comando prende come argomenti i nomi di una serie di file, e produce in uscita un file diviso in colonne, separate dal carattere di tabulazione, in cui i campi di una riga sono costituiti dalle corrispondenti righe dei file in ingresso. Se non si specifica nessun file il comando legge dallo *standard input*, che può essere usato anche all'interno di una sequenza di file indicandolo con "`-`".

Quando i file hanno dimensioni diverse il file prodotto sarà esteso alla lunghezza, in righe, del più lungo dei file in ingresso, in cui le righe finali avranno dei campi vuoti in corrispondenza alle righe mancanti nei file più corti.

Con l'opzione `-s` invece `paste` può effettuare una *trasposizione* dei file, in cui il contenuto (in righe) di ciascuno di essi viene messo in colonna su di un'unica riga. Se si usano più file in ingresso saranno generate tante righe quanti sono i file. Con l'opzione `-d` si possono modificare i caratteri usati per la separazione delle colonne, l'opzione prende come parametro una stringa i cui caratteri saranno usati in sequenza come separatori fra le varie righe, nell'ordine in cui li si sono indicati. Al solito i dettagli si trovano nella pagina di manuale, ma in questo caso sono più esaurienti le informazioni sui comandi generici di Unix disponibili nel sistema degli *info files*, accessibili con `info coreutils` (vedi sez. 2.4.2).

Un altro comando che permette di eseguire una unione per righe è `join`, che però opera solo su due file, che devono essere passati come argomenti, ed esegue l'unione sulla base della presenza di un campo comune, che viene ripetuto una sola volta nell'output finale. In questo caso si può specificare l'uso dello *standard input* indicandolo con "`-`" negli argomenti, ma ovviamente solo per uno dei due file di ingresso.

Il comando opera di default usando come separatori gli spazi, sia per la suddivisione in colonne dei file in ingresso che per la scrittura dei risultati in uscita, utilizzando la prima colonna come campo comune. Quest'ultimo può essere cambiato per entrambi i file con l'opzione `-j`, seguita dal numero della posizione; se il campo comune avesse posizione diversa nei due file si può impostare quest'ultima in maniera separata per ciascuno di essi rispettivamente con le opzioni `-1` e `-2` (sempre seguite dal numero della posizione).

Il comando verifica se in entrambi i file è presente il campo comune e in tal caso scrive in uscita una riga corrispondente, si tenga presente che il campo deve corrispondere esattamente (ma si può usare l'opzione `-i` per richiedere una corrispondenza *case-insensitive*) e che la scansione è eseguita a partire dal primo file una riga alla volta cercando un campo corrispondente nel secondo, e che una volta trovata la corrispondenza la ricerca prosegue, *in entrambi i file*, dalla riga seguente. Questo significa ad esempio che se il campo della prima riga del primo file corrisponde al campo della quarta riga del secondo file, la ricerca di corrispondenza del campo della seconda riga del primo file inizierà a partire dalla quinta riga del secondo, e l'eventuale presenza di una corrispondenza nelle righe precedenti sarà ignorata.

Se ci sono campi vuoti in corrispondenza di un valore questi possono essere riempiti con

Opzione	Significato
`-1` *N*	specifica qual'è il campo comune nel primo file.
`-2` *N*	specifica qual'è il campo comune nel secondo file.
`-j` *N*	specifica qual'è il campo comune.
`-i`	la corrispondenza del campo comune è verificata in modalità *case insensitive*.
`-e` *tag*	specifica un valore da mettere al posto dei campi assenti.
`-t` *c*	specifica un carattere da usare come separatore.

Tabella 2.21: Le principali opzioni del comando `join`.

una stringa fissa indicata come parametro per l'opzione `-e`, e si può cambiare il carattere usato come separatore dei campi usando l'opzione `-t`. L'elenco delle opzioni principali è riportato in tab. 2.21. Al solito si può fare riferimento alla pagina di manuale, ma anche questo caso le informazioni più dettagliate sono disponibili nel sistema degli *info files*, accessibili con `info coreutils`.

Un altro programma molto utile è `sort`, che permette di ordinare il contenuto di un file. Il comando prende come argomento un file e ne stampa il contenuto con le righe in ordine alfabetico. Dato che se non si specifica nessun file il comando opera sullo *standard input*, può essere usato di nuovo in una catena di comandi per riordinare l'uscita di un altro comando. Così se si vuole riordinare un elenco basterà darlo in pasto a `sort`.

Le opzioni di `sort` permettono di controllare le modalità di ordinamento, ad esempio con `-b` si può dire al comando di ignorare gli spazi all'inizio delle righe, con `-r` di invertire l'ordinamento, con `-n` di ordinare le stringhe che contengono numeri sulla base del valore di questi e non di quello alfabetico (per avere 2 prima di 10), con `-f` di non differenziare fra maiuscole e minuscole. Le principali opzioni di `sort` sono riportate in tab. 2.22, per l'elenco completo si faccia al solito riferimento alla pagina di manuale e alla pagina *info files*, accessibile con `info coreutils`.

Opzione	Significato
`-b`	ignora gli spazio ad inizio riga.
`-f`	esegui un ordinamento *case insensitive*.
`-n`	esegue un ordinamento numerico.
`-r`	esegue l'ordinamento in ordine inverso.
`-k` *N,M*	specifica quale colonna usare come chiave.
`-t` *c*	imposta un separatore di campi.
`-u`	elimina le righe duplicate.

Tabella 2.22: Le principali opzioni di `sort`.

In generale `sort` esegue l'ordinamento a partire dall'inizio di ciascuna riga, ma con l'opzione `-k` è possibile selezionare quale sezione della riga usare per l'ordinamento. In questo caso si sottintende che il file sia suddiviso in campi, di default il comando considera i campi separati da un numero arbitrario di caratteri vuoti, come spazi e tabulatori, ma si può specificare un carattere di separazione con `-t`, l'opzione `-k` prevede che si indichino (per valore numerico progressivo) il campo iniziale e quello finale separati da una virgola, ma il campo finale (e la virgola di separazione) possono essere omessi nel qual caso `sort` userà tutto il resto della riga. Così se ad esempio si ha in `nominativi.txt` un elenco contenente i riferimenti di un gruppo di persone, con

nome, cognome e numero di telefono, si potrà ottenerne una versione ordinata per cognome con il comando:

```
sort -k 2,2 nominativi.txt
```

Un altro comando che permette di operare sul contenuto di un file è `uniq`, che elimina linee adiacenti uguali. Il comando prende come argomento un nome di file, ma se non viene specificato niente legge lo *standard input*, e stampa il risultato, senza linee duplicate, sullo *standard output*; usando due argomenti il secondo viene usato come file di destinazione al posto dello *standard output*.

Al solito varie opzioni permettono di controllare le modalità di funzionamento del comando: con `-i` si può ignorare la differenza fra maiuscole e minuscole, con `-s` si può specificare il numero di caratteri ad inizio riga da non inserire nel confronto, ed infine con `-d` si può chiedere ad `uniq` di stampare soltanto le linee duplicate. Altre opzioni, al solito dettagliate nella pagina di manuale, permettono anche selezioni più complesse.

Di nuovo considerata a se stante, l'utilità di un comando come questo può apparire limitata, ma basta pensare alle combinazioni con altri comandi per apprezzarne la funzionalità. Si consideri ad esempio la necessità di riunire elenchi di parole contenuti in più file (supponiamo siano `elenco1.txt`, `elenco2.txt`, ecc.). Lo scopo è quello di avere un file con l'elenco completo in cui tutte le parole compaiono una volta sola; questo può essere ottenuto in un batter d'occhio con un comando come:[24]

```
cat elenco*.txt | sort | uniq > elencofinale
```

Un altro comando che si usa per manipolare il contenuto di un file è `tr`, il cui nome sta per ***tr****anslate*. Lo scopo del comando è quello di effettuare delle sostituzioni dei caratteri. Come tutti i comandi esso opera direttamente su un file qualora questo sia specificato a riga di comando altrimenti utilizza lo *standard input*, scrivendo il risultato sullo *standard output*; in tal modo lo si può utilizzare come un *filtro* di conversione.

Il comando può prendere uno o due argomenti. Il primo indica un insieme di caratteri di cui eseguire la ricerca nel file, il secondo, quando presente, indica la lista dei caratteri con cui il corrispondente carattere del primo insieme deve essere sostituito. In genere quando si specificano due argomenti questi devono specificare due insiemi almeno della stessa dimensione. Qualora questo non avvenga il secondo insieme viene esteso ripetendone l'ultimo carattere, a meno di non usare l'opzione `-t` che tronca invece il primo insieme alle dimensioni del secondo. Se invece il secondo insieme è più lungo vengono utilizzati solo i caratteri iniziali.

Qualora si specifichi l'opzione `-d` il secondo insieme viene ignorato, e tutti i caratteri del primo insieme vengono cancellati. Se invece si specifica l'opzione `-c` il primo insieme viene considerato come quello dei caratteri che *non* devono corrispondere. Infine usando l'opzione `-s` si possono "*strizzare*" preventivamente le ripetizioni di un qualunque carattere del primo insieme trasformandole in un carattere singolo (che poi potrà essere sostituito). L'elenco delle opzioni principali è riportato in tab. 2.23.

Le liste dei caratteri in genere si possono esprimere direttamente con delle stringhe, il comando però supporta anche le classi di caratteri definite in tab. 2.27, e si possono usare gli

[24]in realtà `sort` consente anche di rimuovere le righe duplicate con l'opzione `-u` e l'uso finale di `uniq` in questo caso non servirebbe.

Opzione	Significato
`-c`	seleziona i caratteri che non sono indicati nel primo insieme.
`-d`	cancella i caratteri indicati nel primo insieme.
`-s`	rimpiazza le occorrenze ripetute dei caratteri del primo insieme.
`-t`	tronca il primo insieme alla lunghezza del secondo.

Tabella 2.23: Le principali opzioni di `tr`.

identificatori ivi riportati. Inoltre l'utilizzo del carattere di *escape* "\" permette non solo di proteggere i caratteri speciali, ma di inserire negli insiemi caratteri non stampabili secondo quanto riportato in tab. 2.24. Per la sintassi completa del comando ed i dettagli riguardo le varie forme che si possono utilizzare per specificare gli insiemi dei caratteri si faccia comunque riferimento alla pagina di manuale, al solito accessibile con `man tr`.

Espressione	Significato
`\NNN`	carattere specificato col suo valore numerico ottale.
`\\`	barra trasversa.
`\a`	*bell* (suona sul terminale).
`\b`	cancella indietro (*backspace*).
`\f`	pagina nuova (*form feed*).
`\n`	a capo (*new line*).
`\r`	ritorno carrello (*return*).
`\t`	tabulazione orizzontale.
`\v`	tabulazione verticale.

Tabella 2.24: Caratteri speciali ad uso del comando `tr`.

Analoghi a `tr`, ma con uno scopo molto più limitato, sono i due programmi `expand` e `unexpand` che servono rispettivamente a convertire tabulazioni in spazi e viceversa. Entrambi prendono l'opzione `-t`. Questa, se seguita da un numero singolo, permette di specificare le dimensioni del tabulatore, mentre si possono impostare le diverse posizioni delle tabulazioni se il parametro viene specificato come lista di numeri separata da virgole. Per i dettagli di nuovo si può fare riferimento alle rispettive pagine di manuale.

Un'altra serie di programmi attinenti l'elaborazione del contenuto dei file sono quelli relativi a compressione e decompressione degli stessi. Nei primi sistemi Unix veniva usato a questo scopo il comando `compress`, una alternativa, nata sotto DOS, è invece `zip`. Entrambi i programmi sono stati portati su Linux, anche se con alcune limitazioni dovute alla presenza di brevetti software.

Per questo motivo il comando di compressione più usato è `gzip`, creato dal progetto GNU per sostituire i due precedenti. La cosa però è possibile al 100% soltanto per `compress` che infatti viene normalmente sostituito con `gzip`, il problema con `zip` si presenta in caso di archivi di file, perché `gzip` è in grado di operare soltanto su file singoli, per gestire archivi su Unix infatti si usano altri programmi (vedi sez. 4.1.2), per cui in caso di archivio creato con `zip` si dovrà utilizzare il comando `unzip`.

Se utilizzato senza opzioni il comando richiede come argomenti la lista dei file da comprimere, una volta completata la compressione ciascun file verrà sostituito dalla versione compressa, riconoscibile per il suffisso `.gz`, mantenendo tutte le altre proprietà del file originale (tempi, permessi e proprietario), qualora il nuovo file già esistesse il programma si blocca chiedendo se deve sovrascriverlo.

La decompressione di un file può essere effettuata invocando il comando con l'opzione `-d`, ma è molto più comune eseguirla invocando il comando stesso come `gunzip`, è comune infatti avere `gunzip` come *hard link* a `gzip`, anche se alcune distribuzioni preferiscono usare uno script di shell per lanciare `gzip -d`. In questo caso il file `.gz` verrà decompresso, ed una volta riottenuto il file originale, cancellato.

Una ulteriore modalità con cui si può invocare il comando è come `zcat`, che consente di decomprimere un file inviando il risultato sullo *standard output*, così da poterne utilizzare i contenuti con una *pipeline*. In questo caso il comando è identico all'invocazione di `gzip -d -c`, dove l'opzione `-c` consente di inviare il risultato del comando sullo *standard output*, e vale anche in questo caso quanto detto per `gunzip`, per cui anche `zcat` può essere installato come *hard link* a `gzip`. Le principali opzioni del comando sono riportate in tab. 2.25, per tutti i dettagli si consulti al solito la pagina di manuale.

Opzione	**Significato**
`-c`	invia i risultati sullo *standard output*.
`-d`	decomprime anziché comprimere.
`-l`	stampa le caratteristiche dei file compressi.
`-t`	controlla l'integrità dei file compressi.

Tabella 2.25: Le principali opzioni di `gzip`.

Benché `gzip` sia un programma di compressione abbastanza efficiente e veloce, l'algoritmo utilizzato non è uno dei più potenti, per questo motivo, se si vogliono ottenere delle compressioni maggiori, è disponibile un altro programma, `bzip2`, che utilizza un diverso algoritmo di compressione più efficiente ma anche molto più lento. In questo modo è possibile ottenere una compressione maggiore a scapito di un tempo di elaborazione per la compressione dei file più lungo.

A parte l'algoritmo di compressione, e l'uso del suffisso `.bz2` per i file compressi, le differenze con `gzip` sono minime: le opzioni `-c`, `-d` e `-t` sono identiche, e sono disponibili come alternative a `zcat` e `gunzip` gli equivalenti `bzcat` e `bunzip2`. Anche in questo caso per i dettagli si può fare riferimento alla pagina di manuale.

Un ulteriore programma di compressione ancora più sofisticato è `xz` basato sull'algoritmo LZMA, che nasce come evoluzione del precedente `lzma`, di cui rappresenta anche un sostituto. Presenta ottimi valori di compressione e velocità rispetto a `bzip2`, ma comporta anche un consumo più elevato di memoria, soprattutto in fase di compressione, che sulle macchine moderne è meno critico ma può diventare problematico su macchine con risorse molto limitate. Sta riscuotendo un discreto successo e molti progetti, vista la migliore compressione ottenibile anche rispetto a `bzip2`, stanno iniziando ad adottarlo nella distribuzione di archivi compressi.

Il comando supporta, analogamente a `gzip`, l'uso di opzioni nella forma `-N` dove `N` varia da 0 a 9 ed indica un livello di compressione; il default usato è `-6`, che è una media fra occupazione di memoria necessaria per la decompressione, e livello di compressione. Dato che l'uso della memoria può essere critico (se non presente in quantità sufficiente la decompressione fallisce), è possibile impostare un limite massimo che non deve essere superato con `--memlimit`, che in fase di compressione fa si che il livello della stessa sia automaticamente diminuito per garantire che il limite indicato non sia mai superato nel decomprimere il file.

I file prodotti da xz sono nel nuovo formato LZMA2 ed usano normalmente l'estensione .xz; come per gzip e bzip2 sono disponibili una serie di comandi ausiliari, tutti equivalenti all'uso di xz con le opportune opzioni: per decomprimere un file si usa unxz, e per scrivere sullo *standard output* il contenuto decompresso si usa xzcat. Il programma supporta anche la sintassi del precedente lzma per gestire i file nel formato di quest'ultimo,[25] per cui lo si può invocare anche come lzma, unlzma e lzcat.

Trattiamo infine un terzo insieme di programmi che operano su file di testo principalmente a scopo di riformattarne adeguatamente il contenuto. Il primo di questi è pr che serve a suddividere il testo in pagine, numerate progressivamente dal programma stesso, in un formato adatto alla stampa (da cui deriva il nome); il programma supporta numerose opzioni volte a configurare in maniera diversa le modalità con cui viene effettuata la suddivisione in pagine ed impostare le informazioni mostrate in testa e coda delle stesse.

Il secondo programma è fmt che riformatta il file in uscita paragrafo per paragrafo, permettendo di impostare caratteristiche come indentazioni della prima riga e numero di spazi e mantenere una lunghezza costante della riga. Di nuovo le varie modalità con cui si può compiere la formattazione sono controllate dalle opzioni, per le quali si rimanda alla pagina di manuale.

Il terzo programma è fold che si limita invece a spezzare le righe troppo lunghe, inserendo degli a capo, rispetto ad una dimensione fissa da assegnare alle stesse, specificabile con l'opzione -w. Questo è il più semplice dei comandi di formattazione, con forse l'eccezione di nl, che, come indica il nome, serve a numerare le linee di ciascun file. Di nuovo si può fare riferimento alle rispettive pagine di manuale per le singole opzioni.

2.2.5 Ricerche ed elaborazioni sui file attraverso le espressioni regolari

Abbiamo visto in sez. 2.2.2 come find ci permetta di trovare un file in base al nome e alle sue proprietà, in sostanza ci si è limitati al nome e alle altre caratteristiche mantenute nell'*inode*. Non abbiamo parlato però di come effettuare ricerche all'interno dei file stessi in base al loro contenuto. Il comando che fornisce questa funzionalità è grep, che usato in forma elementare serve a cercare una stringa di caratteri all'interno di uno o più file; la potenza di questo comando (e delle varie versioni *evolute*, come egrep) è dovuta al fatto che consente di fare ricerche estremamente evolute attraverso l'uso delle cosiddette *espressioni regolari*.

Le *espressioni regolari*, abbreviate spesso in *regex*, dal loro nome in inglese, *regular expression*, sono una specie di estensione del sistema del *filename globbing* che abbiamo illustrato in sez. 2.1.3, in cui, attraverso una serie di operatori, si possono cercare corrispondenze fra stringhe con un grado di complessità incredibilmente elevato. Questo le rende allo stesso tempo uno degli strumenti più potenti ed uno degli argomenti più ostici del mondo Unix.

L'uso elementare di grep è banale, il comando prende come primo argomento la stringa da cercare seguita dal nome del file, o dei file se se ne indica più di uno, in cui effettuare la ricerca. Il comando stampa in uscita ogni riga del file nella quale ha rilevato una corrispondenza, e se si sono indicati più file precede la riga con il nome del file. Ad esempio se si cerca la stringa "*Dispense*" in alcuni file di questa guida si otterrà:

```
piccardi@anarres:~/truedoc/corso$ grep Dispense net*.tex
netadmin.tex:%% Dispense Amministrazione di rete
```

[25] in sostanza l'invocazione di lzma è equivalente ad eseguire xz -format=lzma.

```
netbase.tex:%% Dispense Amministrazione di rete
netdns.tex:%% Dispense Amministrazione di rete
netinter.tex:%% Dispense Amministrazione di rete
netlpi.tex:%% Dispense Amministrazione di rete
```

Altre opzioni del comando sono `-i` che permette di effettuare ricerche *case insensitive*, `-r` che effettua la ricerca ricorsivamente qualora nell'elenco dei file si sia indicata una directory, e `-v` che inverte il risultato della ricerca (cioè stampa le righe che *non* corrispondono alla stringa utilizzata). Le opzioni del comando sono moltissime ma si sono riportate le più importanti in tab. 2.26; al solito si rimanda alla pagina di manuale per una descrizione più completa.

Opzione	**Significato**
`-b`	stampa la posizione nel file (in byte) in testa a ciascuna riga in output.
`-c`	non stampa le righe con le corrispondenze, ma solo il numero totale delle stesse.
`-E`	interpreta la stringa di ricerca come una espressione regolare estesa.
`-e`	indica esplicitamente la stringa di ricerca (serve a proteggere stringhe di ricerca che iniziano con "`-`").
`-G`	interpreta la stringa di ricerca come espressione regolare normale (il default).
`-H`	stampa il nome del file in testa a ciascuna riga di output (anche se si ricerca su un solo file).
`-h`	sopprime la stampa del nome del file in testa a ciascuna riga di output (quando sono più di uno).
`-i`	non distingue fra maiuscole e minuscole.
`-m` *`N`*	sospende la stampa in uscita dopo *`N`* corrispondenze.
`-n`	stampa il numero di riga del file in testa a ciascuna linea in uscita.
`-P`	interpreta la stringa di ricerca come espressione regolare in stile Perl.
`-r`	esegue la ricerca in forma ricorsiva, ripetendola anche per tutti i file contenuti nelle sottodirectory di quelle passate in ingresso (può essere specificata anche come `-R`).
`-s`	sopprime la stampa degli errori al riguardo di file non leggibili o non esistenti.
`-v`	inverte la selezione, stampa le linee non corrispondenti.
`--color`	colora diversamente il testo corrispondente (disponibile solo nelle versioni più recenti di GNU `grep`).

Tabella 2.26: Principali opzioni del comando `grep`.

Come molti altri comandi anche `grep` legge, qualora non gli si sia passato nessun file come argomento, dallo *standard input*, i risultati al solito vengono scritti sullo *standard output*. Diventa allora evidente la sua utilità come filtro per selezionare a piacere, sulla base delle opportune corrispondenze, le righe di un file. Si noti come facendo così si possano effettuare ricerche sempre più mirate semplicemente concatenando in successione diverse chiamate al comando.

Come accennato la vera potenza di `grep` sta nel fatto che la ricerca non viene semplicemente eseguita sulla corrispondenza ai caratteri contenuti nella stringa passata come argomento, ma nel fatto che quest'ultima viene interpretata, attraverso l'uso di alcuni caratteri riservati, come

un *pattern* all'interno del file, indicato con una speciale sintassi che è quella che costituisce le citate *espressioni regolari*.

La trattazione dettagliata delle espressioni regolari va ben al di là delle possibilità di questo testo,[26] qui ci limiteremo a fornire la descrizione delle funzionalità più elementari. Una espressione regolare è in sostanza una stringa di caratteri che identifica un *pattern*, cioè una struttura ordinata di caratteri e stringhe, di cui cercare l'occorrenza in un file eseguendo l'operazione che viene usualmente chiamata *pattern matching*.

In maniera analoga ad una espressione matematica una espressione regolare viene costruita combinando delle espressioni elementari attraverso gli opportuni operatori. Gran parte dei caratteri (tutte le lettere, maiuscole o minuscole, e i numeri) di una espressione regolare non viene interpretata, e la loro presenza richiede semplicemente la presenza di un corrispondente carattere nel contenuto su cui si effettua la ricerca, alcuni caratteri sono invece riservati per svolgere il ruolo di operatori; uno di questi è la barra rovescia "\" con la quale si può richiedere l'interpretazione letterale del carattere successivo (bloccandone l'interpretazione come operatore).

Il carattere "`^`" viene utilizzato per identificare l'inizio di una riga, mentre il carattere "`$`" serve ad identificarne la fine; pertanto si potranno identificare i commenti in uno script di shell con l'espressione regolare "`^#`" mentre con l'espressione "`^$`" si identificheranno le righe vuote. Il carattere "`.`" viene invece utilizzato nelle espressioni regolari per indicare un carattere qualunque.

Classe	**Significato**
`[:alnum:]`	lettere (maiuscole e minuscole, indipendenti dalla localizzazione) e numeri.
`[:alpha:]`	lettere (maiuscole e minuscole, indipendenti dalla localizzazione).
`[:cntrl:]`	caratteri di controllo.
`[:digit:]`	caratteri numerici.
`[:graph:]`	caratteri stampabili (esclusi gli spazi).
`[:lower:]`	lettere minuscole.
`[:print:]`	caratteri stampabili (caratteri vuoti compresi).
`[:punct:]`	caratteri di punteggiatura.
`[:space:]`	caratteri vuoti verticali ed orizzontali (spazi, tabulatori, ritorni a capo).
`[:upper:]`	lettere maiuscole.
`[:xdigit:]`	cifre esadecimali.

Tabella 2.27: Le classi di caratteri utilizzabili nelle espressioni regolari.

Alcune delle espressioni usate nel *filename globbing* di sez. 2.1.3 si ritrovano anche nelle espressioni regolari, anche se in questo caso assumono un significato leggermente diverso e per questo sono spesso causa di confusione ed errori. In particolare le parentesi quadre vengono utilizzate come nel *filename globbing* per indicare una lista o un intervallo di caratteri (ad esempio "`[aeiou]`" indica le vocali e "`[a-z]`" le minuscole), la differenza col *filename globbing* è che in questo caso il carattere "`^`" messo all'inizio viene interpretato come inversione della lista seguente (ad esempio con "`[^A-Z]`" si indicano tutti i caratteri che non siano una lettera maiuscola). Oltre a liste e intervalli specificati direttamente si possono indicare fra parentesi quadre una serie di *classi* predefinite di caratteri con le espressioni riportate in tab. 2.27.

Anche i caratteri "`*`" e "`?`" assumono un significato simile a quello del *filename globbing*, ma nel caso delle espressioni regolari questo accade perché essi vanno a far parte di un gruppo particolare di operatori che sono chiamati *operatori di ripetizione*, riportati in tab. 2.28. Gli

[26]un buon testo sull'argomento è [RegExp].

operatori di ripetizione si applicano all'oggetto che li precede (che sia un carattere singolo o una espressione complessa) e richiedono che l'occorrenza dello stesso avvenga un certo numero di volte, nel caso di "`*`" questo può essere un numero qualunque (compreso nessuna) mentre con "`?`" si richiede che sia al più una volta (e di nuovo lo zero è compreso). Spesso risulta utile l'uso dell'operatore "`+`" che richiede la presenza di almeno una volta.

Operatore	**Significato**
`?`	l'espressione precedente può corrispondere zero o una volta.
`*`	l'espressione precedente può corrispondere da zero ad un qualsiasi numero di volte.
`+`	l'espressione precedente può corrispondere da uno ad un qualsiasi numero di volte.
`{n}`	l'espressione precedente deve corrispondere esattamente `n` volte.
`{n,}`	l'espressione precedente deve corrispondere `n` o più volte.
`{n,m}`	l'espressione precedente deve corrispondere fra `n` e `m` volte.

Tabella 2.28: Gli *operatori di ripetizione* nelle espressioni regolari.

Questo significato è di nuovo leggermente diverso da quello presente nel *filename globbing*, in particolare una espressione come "`ab*`" nel primo caso seleziona tutti i file il cui nome inizia per "`ab`" seguito da un numero qualunque di altri caratteri qualsiasi, mentre nel secondo caso corrisponde ad una linea che contenga una "`a`" seguita da un numero qualunque (compreso zero) di "`b`", per cui anche una stringa come "`ac`" corrisponderebbe. Per riottenere lo stesso significato precedente con una espressione regolare occorrerebbe usare la stringa "`ab.*`", in questo modo si richiede la presenza iniziale di entrambe le lettere "`a`" e "`b`" seguite da un qualunque numero (indicato da "`*`") di un qualunque altro carattere (indicato da "`.`").

Negli esempi appena mostrati gli operatori di ripetizione sono applicati solo al carattere che li precede, è possibile però applicarli ad una intera espressione regolare mettendola fra parentesi tonde, usando quello che viene chiamato un *raggruppamento*. Ad esempio per trovare in questo testo gli errori di battitura in cui si era scritto due volte la parola `una` si è usato il comando:

```
piccardi@hain:~/truedoc/corso$ grep -E '(una )2,' *.tex
shell.tex:mentre nel secondo caso corrisponde ad una una linea che contenga una
```

dove appunto si è richiesto che l'occorrenza della stringa "`una `" avvenisse almeno due volte di fila.

Un raggruppamento viene a costituire quello che viene chiamato un *subpattern*; e non solo si può richiedere, con l'uso degli operatori di ripetizione la presenza multipla di uno stesso *pattern*, ma si può anche usare l'operatore "`|`" posto fra due raggruppamenti per richiedere la presenza alternativa di uno di essi. Così si può richiedere la presenza della parola `free` o della parola `copyleft` nella stessa riga con:

```
piccardi@hain:~$ grep '\(free\|copyleft\)' /usr/share/common-licenses/GPL-3
  The GNU General Public License is a free, copyleft license for
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
...
```

In questo caso però si ha una complicazione e la sintassi varia se si usa la sintassi delle cosiddette *espressioni regolari semplici*, il default, nella quale le parentesi tonde e la "`|`" devono

essere protetti all'interno dell'espressione regolare stessa con la barra trasversa "\". Si noti infatti come si sia anche espressa quest'ultima fra degli apici singoli per bloccare l'interpretazione degli stessi caratteri da parte della shell. Se invece si usa la sintassi delle *espressioni regolari estese* (attivate con l'opzione `-E` di `grep`) la protezione all'interno dell'espressione non è necessaria, ma va comunque mantenuta quella nei confronti dell'interpretazione da parte della shell.

L'uso dei *subpattern* ha poi un'ulteriore vantaggio, e cioè che tutte le volte che uno di questi viene trovato, il suo valore viene memorizzato in un elenco progressivo (secondo la sua posizione nell'espressione regolare) e detto valore può essere referenziato (e riutilizzato all'interno della stessa espressione) con l'uso delle sequenze speciali "`\1`", "`\2`", ecc. Così se ad esempio si vogliono trovare tutte le righe in cui compare due volte, in posizione qualsiasi, una stessa identica combinazione di almeno 10 caratteri si potrà utilizzare l'espressione:

```
piccardi@hain:~$ grep -E '(.{10,}).*\1' /usr/share/common-licenses/BSD
3. Neither the name of the University nor the names of its contributors
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
```

dove fra parentesi si è richiesta la selezione di 10 caratteri consecutivi qualunque e poi si è richiesto che la stessa selezione di caratteri comparisse di seguito sulla stessa linea (con interposti un numero qualunque di altri caratteri).

Come si può notare la selezione ottenuta può non essere immediata da capire, con le ultime versioni di `grep` viene però in aiuto l'opzione `--color` che permette di stampare in un colore diverso la sezione della riga che corrisponde, così da rendere più evidente l'effetto dell'espressione regolare, cosa che può risultare di grande aiuto quando si eseguono selezioni complesse e ci si trova a che fare con risultati inaspettati.

Se `grep` utilizza le espressioni regolari per eseguire delle ricerche sui contenuti di un file, consentendo selezioni molto complesse, un altro comando, `sed`, il cui nome sta per ***s**tream **ed**itor* le usa per effettuare manipolazioni del contenuto. Il programma serve per eseguire una serie di trasformazioni su un flusso di dati, analoghe a quelle che potrebbero essere fatte usando un normale editor su un file. Questo consente di poter utilizzare il comando in una catena di *pipe* che mettano in grado di effettuare modifiche anche molto complesse nel passaggio dei dati dallo *standard input* allo *standard output*.

Il comando prende come primo argomento una *espressione di editing* che indica le operazioni da eseguire, a meno che non la si sia specificata nel contenuto del file indicato con l'opzione `-f` o come parametro per l'opzione `-e`. I restanti argomenti, se presenti, indicano i file su cui applicare la suddetta espressione; se non è presente nessun altro argomento lo script viene applicato sullo *standard input*. Le principali opzioni sono riportate in tab. 2.29.

A meno di non utilizzare l'opzione `-n` il comando stampa tutto quanto è stato letto in ingresso con le eventuali modifiche apportate sullo *standard output*. È così possibile salvare il risultato del filtraggio redirigendo l'output del comando. L'opzione `-n` è fornita per permettere, all'interno dei vari comandi, di eseguire una stampa in uscita solo quando lo si ritiene opportuno, ed ottenere pertanto un risultato diverso. Con l'opzione `-i` si può sovrascrivere il file "*sul posto*", evitando una copia esterna,[27] col rischio di perdere il contenuto precedente, per questo l'opzione supporta la possibilità di specificare un parametro che indichi il suffisso da apporre al nome del file per ottenere un backup della versione originale.

[27] si ricordi che redirigere lo *standard output* su un file ne comporta il troncamento, se lo si facesse sul file che si sta modificando di potrebbe incorrere nelle conseguenze sottolineate in sez. 2.1.4.

Opzione	Significato
`-e` *expr*	indica esplicitamente che l'argomento seguente è l'espressione contenente i comandi di editing da applicare.
`-f` *file*	indica un file contenente l'espressione dei comandi di editing.
`-n`	sopprime la stampa del *pattern space* che avviene alla fine del processo di ogni riga.
`-r`	usa le espressioni regolari estese.
`-i [`*suf*`]`	effettua la sostituzione sovrascrivendo direttamente il file, eventualmente creando un backup se si è specificato il suffisso *suf* opzionale.

Tabella 2.29: Principali opzioni del comando `sed`.

La potenza di `sed` sta nella grande flessibilità dei comandi che possono essere dati tramite l'espressione di editing, alcuni di questi infatti utilizzano le espressioni regolari per individuare dei *pattern* nel file, ed operare su di essi compiendo sostituzioni, cancellazioni, ecc.

Le espressioni di editing sono normalmente divise in *indirizzi* e *comandi*. Un indirizzo esprime l'insieme delle righe del file originario a cui si applica il successivo comando. Un singolo indirizzo indica la riga cui si applica il comando, con due indirizzi separati da una virgola si indica un intervallo di righe, mentre quando, come nella maggior parte dei casi, non si specifica nessun indirizzo, il comando si applica a tutte le righe del file. Infine se ad un indirizzo si fa seguire (prima del comando) il carattere "`!`" la selezione delle righe viene invertita.

In genere un indirizzo viene specificato tramite il numero della relativa riga (ad esempio "`10`" indica la riga 10, "`5,15`" indica le righe dalla quinta alla quindicesima e "`3,7!`" tutte le righe escluse quelle nell'intervallo fra la terza e la settima. Sono però possibili delle selezioni più complesse, come "`$`" che indica l'ultima riga del file, o "`/regexp/`" che seleziona le righe corrispondenti ad una espressione regolare messa fra le due barre; se usato in questo modo, senza dare nessun comando, `sed` agisce in maniera sostanzialmente identica a `grep`. Un elenco delle principali forme con cui si può specificare un indirizzo è riportato in tab. 2.30.

Espressione	Significato
`N`	selezione la `N`-sima riga.
`N,M`	seleziona le righe dalla `N` alla `M`.
`N~M`	selezione le righe a partire dalla `N`-sima a passi di `M`.
`/regexp/`	seleziona le righe che corrispondono all'espressione regolare `/regexp/`.
`$`	seleziona l'ultima riga del file.
`N,+M`	seleziona `M` righe a partire dalla `N`-sima.

Tabella 2.30: Espressioni per specificare un indirizzo al comando `sed`.

Una volta specificato l'*indirizzo* la seconda parte di una espressione di editing è composta da un *comando*. Un comando viene sempre introdotto da un carattere di controllo, seguito da eventuali parametri. I comandi possono essere ripetuti e raggruppati con l'uso delle parentesi graffe, ed in genere vengono scritti uno per riga e possono essere letti da un file con l'opzione `-f`.

Per capire l'azione dei vari comandi occorre capire come `sed` effettua la manipolazione dei dati e dove questi sono mantenuti. Quando una riga viene letta da un file essa viene posta nel cosiddetto *pattern space* dove vengono effettuate tutte le operazioni e la manipolazione del

contenuto: è cioè nel *pattern space* che si troveranno le modifiche eseguite al contenuto originale dalle operazioni di `sed`. Il comando prevede inoltre la presenza di un altro spazio, chiamato *hold space*, che inizialmente è vuoto, e dove è possibile inviare dati a partire dal *pattern space* per poterli riprendere successivamente; questo permette operazioni complesse in cui i dati sono opportunamente spostati e ricombinati all'interno di detti *spazi*.

L'elenco dei comandi più comuni è riportato in tab. 2.31 ma `sed` ne supporta molti altri. Un elenco completo con una descrizione sommaria si può trovare nella pagina di manuale, ma la documentazione completa, fornita anche di parecchi esempi, è disponibile solo nelle pagine *info*, accessibili con `info sed`.

Espressione	**Significato**
`q` *`N`*	esce con codice di uscita pari a *`N`*.
`p`	stampa il *pattern space*.
`d`	cancella il *pattern space* e passa al ciclo seguente.
`s`	sostituisce il testo corrispondente ad una espressione regolare con un altro testo.
`p`	stampa il *pattern space*.
`i`	inserisce una riga al di sopra di quella selezionata.
`a`	aggiunge una riga dopo quella selezionata.

Tabella 2.31: Principali espressioni di comando per `sed`.

Il più importante ed utilizzato, e l'unico che tratteremo esplicitamente, dei comandi di `sed` è "`s`" che permette di sostituire una sezione di testo nel *pattern space* con un'altra. La sintassi del comando è nella forma `s/ricerca/sostituzione/` dove al posto del carattere "`/`" si può usare un carattere qualsiasi che viene ad assumere il ruolo di delimitatore, questo consente di usare un altro carattere quando nella stringa di ricerca o di sostituzione appare la stessa "`/`" (ad esempio in caso di *pathname*). In questa forma elementare il comando rimpiazza (all'interno del *pattern space*) la prima occorrenza della stringa `ricerca` con la stringa `sostituzione`.

La potenza del comando sta nel fatto che la stringa di ricerca è in realtà una espressione regolare, pertanto diventa possibile fare selezioni estremamente complesse; inoltre si possono utilizzare i *subpattern* già visti con `grep` per selezionare pezzi di testo che possono essere riutilizzati nella stringa di sostituzione con le usuali espressioni `\1`, `\2`, ecc. consentendo così manipolazioni molto sofisticate.

Dopo la "`/`" finale si possono specificare degli ulteriori sottocomandi, ad esempio usando "`g`" si indica al comando di sostituire tutte le occorrenze della stringa di ricerca, e non solo la prima, con "`p`" si richiede la stampa del *pattern space* (si usa in genere in combinazione con l'opzione `-n` per far stampare solo le righe modificate) mentre specificando un numero "`N`" si esegue la sostituzione solo per la `N`-sima corrispondenza trovata, con "`i`" invece si richiede che la ricerca sia *case-insensitive*.

2.3 Gli editor di testo

Si è preferito mantenere in una sezione separata la trattazione di una classe di programmi, quella degli *editor* di testo, che vengono a costituire uno dei principali strumenti usati da tutti gli amministratori di sistema. Lo scopo di questa sezione è quello di mettere il lettore in grado di

cavarsela con i principali editor disponibili in tutte le distribuzioni, poi con il tempo e l'esperienza ognuno finirà con l'adottarne uno come preferito.

2.3.1 Introduzione

Una delle caratteristiche fondamentali di un sistema unix-like, per le ragioni che tratteremo in dettaglio in sez. 3.1.1, è quella di mantenere le configurazioni dei programmi all'interno di semplici file di testo. Per questo motivo probabilmente il programma più utilizzato dai professionisti nell'amministrazione di sistema per configurare i servizi è l'*editor* di testi.

Infatti anche se la disponibilità di strumenti che permettono le più comuni operazioni di amministrazione tramite una interfaccia grafica sta crescendo, questi in genere mettono a disposizione solo un limitato numero di opzioni, e non danno mai il completo controllo sul comportamento di un programma, che si può ottenere soltanto operando direttamente sui file di configurazione. Per questo motivo l'*editor*, cioè un programma che permetta di leggere file di testo e modificarne il contenuto, diventa il principale strumento dell'amministrazione di sistema.

Inoltre quando un qualche problema sul disco, o il classico `rm -fR` dato da *root* un po' troppo allegramente, avrà danneggiato qualche file essenziale o bloccato il sistema all'avvio, sarà sempre possibile usare una distribuzione di recupero avviata da CD o da chiavetta USB dove gli strumenti grafici potrebbero non essere disponibili. E quand'anche si usasse una delle tante distribuzioni che sono in grado di fornire un desktop completo, le configurazioni sul sistema danneggiato andrebbero comunque effettuate a mano dato che anche se il sistema fosse dotato degli strumenti grafici di amministrazione necessari, questi opererebbero sul sistema avviato, non su quello danneggiato.

Infine, anche se la diffusione della banda larga riduce il problema, usare una interfaccia grafica da remoto resta sempre, considerazioni di sicurezza a parte, estremamente lento (e sostanzialmente impossibile senza una buona ADSL), mentre con la riga di comando si può usare un terminale remoto e fare tutto quello che si vuole anche con la banda fornita da un semplicissimo modem analogico.

Dato che l'editor di testi ha sempre avuto questo ruolo fondamentale, è stato uno dei principali programmi applicativi, sviluppato fin dagli esordi di Unix. Per questo, come per molte altre applicazioni di uso generale, ne esistono molte versioni, con i nomi più pittoreschi, che pur svolgendo lo stesso compito di base (la modifica dei file di testo) vanno da quelli dotati solo delle funzionalità più elementari, come `ed`, che permette di operare solo su una linea alla volta,[28] ai più complessi e sofisticati, come `emacs` che ha una quantità infinita di funzionalità diverse e viene paragonato da alcuni esagerati ad un secondo sistema operativo.

Nel prosieguo di questa sezione daremo una breve panoramica sull'uso dei più comuni editor di testo, ma restando nell'ottica dell'amministrazione di sistema tratteremo esclusivamente di editor accessibili da *console*, e quindi utilizzabili anche attraverso connessioni remote con dei semplici modem. Non entreremo nei dettagli dell'uso dei singoli editor, ci limiteremo a esporre quali sono i comandi base per leggere, modificare e scrivere su un file di testo.

La scelta dell'editor resta comunque una scelta personale, che genera spesso clamorose "*guerre di religione*" fra le fazioni dei sostenitori dei diversi editor. Particolarmente virulente sono quelle

[28] è probabilmente il primo editor sviluppato in ambiente Unix; per questa caratteristica viene chiamato "*editor di linea*", ed ha origine ai tempi in cui i terminali erano delle telescriventi; benché oggi nessuno sia così masochista da usarlo direttamente, i suoi comandi si ritrovano in altri editor e pure in programmi come `sed`.

fra i sostenitori di `emacs` e `vi`, i più blasonati (per numero di utenti e tradizione di utilizzo) fra gli editor di testi.

2.3.2 Un editor evoluto: `emacs`

Per molti `emacs` è l'editor. Il suo nome tutt'altro che immediato in teoria starebbe per ***e**xtensible* ***mac**ro **s**ystem*, ma quando la memoria era molto poca è stato ribattezzato in ***e**ight **m**egs **a**nd* ***c**onstantly **s**wapping*, oppure, facendo riferimento all'abbondante uso di tasti di modifica, ***e**scape* ***m**eta **a**lt **c**ontrol **s**hift*.

Sicuramente come editor è il più potente, dentro `emacs` si può davvero fare di tutto: navigare fra i file e in Internet, leggere la posta e le news di *Usenet*, programmare (con evidenziazione della sintassi e scorciatoie ai costrutti principali di qualunque linguaggio), fare debug, scrivere un testo come questo, giocare (a tetrix o a qualche avventura testuale), ed anche farsi psicanalizzare dal "*doctor*".[29]

Qualunque cosa sia possibile fare con del testo, con `emacs` si fa, infatti l'editor è programmabile in un dialetto specifico del Lisp, (un linguaggio funzionale estremamente potente che consente una enorme espandibilità del programma) e per un qualunque compito specifico sono state create le estensioni più varie in grado di eseguire operazioni complesse, gestire automatismi, abbellire la visualizzazione, ecc.

Tutto questo ha ovviamente un costo, ed infatti i detrattori di `emacs` ne lamentano la pesantezza, ma data la diffusione e la potenza dello strumento, il fatto che le distribuzioni su dischetto ormai sono passate di moda, e la mia preferenza personale nei suoi confronti, ne parleremo più estesamente considerato poi che molti altri editor ne hanno copiato la sintassi e le modalità d'uso, ed alcuni programmi forniscono modalità di comando *compatibili*, come la shell, che usa le stesse combinazioni di tasti per l'editing della riga di comando, come abbiamo visto in sez. 2.1.3.

Inoltre sia `emacs`, che il cugino `xemacs`, che nacque proprio per questo, sono editor usabili direttamente anche dall'interfaccia grafica, nel qual caso verranno forniti all'utente gli usuali menu a tendina in cima alla finestra, attraverso i quali si potranno trovare buona parte delle funzionalità di cui essi dispongono.

Come per tutti gli editor una sessione di `emacs` inizia invocando il comando seguito dal nome del file da modificare; in tal caso si otterrà una finestra come quella mostrata in fig. 2.1. Nella prima riga si trova il *menu* dei comandi, ad esso segue la sezione principale, dove compare il testo del file ed in cui ci si muove con le frecce, terminato da una *barra di stato* in cui compaiono varie informazioni (il nome del file, se sono state fatte modifiche, ecc.). Nella riga finale viene tenuto il cosiddetto *minibuffer*, in cui compaiono brevi messaggi (come nell'esempio, che riporta la scritta `(Updating...done)`) ed in cui si scrivono gli argomenti dei comandi più complessi.

Uno dei concetti fondamentali di `emacs` è quello di *buffer*, qualunque porzione di testo venga utilizzata da `emacs`, con l'eccezione del *menu* e della *barra di stato*, è mantenuta in un *buffer*, sia che si tratti di un file che si è aperto per lavorarci sopra, che del testo generato automaticamente nelle operazioni dell'editor (come le liste dei file da scegliere che compaiono quando si usa l'autocompletamento). Anche il *minibuffer* è un buffer, con la peculiarità di essere posizionato sull'ultima riga ed essere utilizzato per l'interazione con i comandi.

In genere un buffer viene visto su una *finestra* che ne mostra il contenuto ed è terminata da una barra di stato (nell'esempio ce n'è solo una), ma all'interno di una stessa istanza dell'editor

[29] si provi ad eseguire `M-x doctor`...

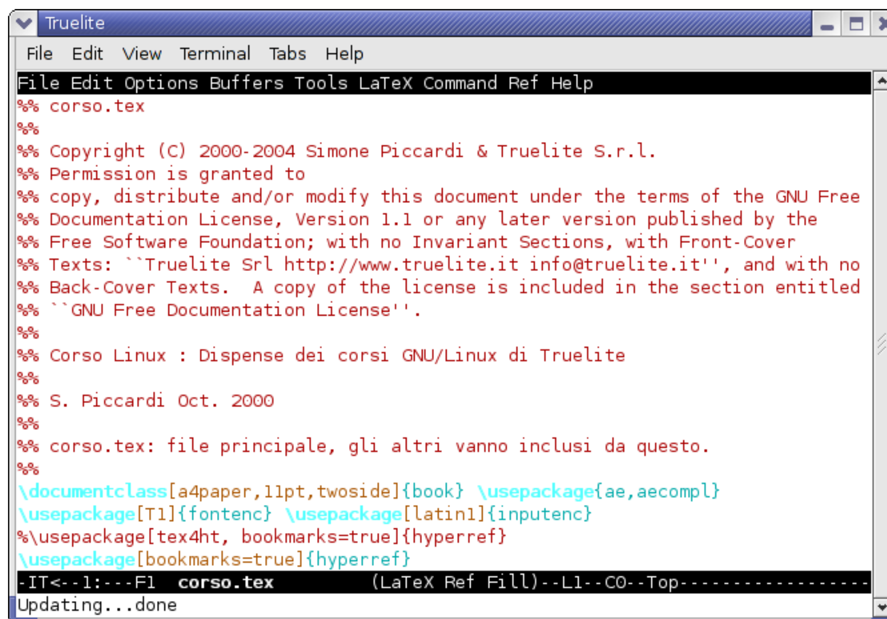


Figura 2.1: Schermata di avvio dell'editor `emacs`.

possono anche esserci più *finestre* che possono essere aperte sullo stesso buffer o su buffer diversi; questo vuol dire che ad esempio la sezione centrale di fig. 2.1 può venire divisa in due. Questo permette, anche operando in *console*, di lavorare su due o più finestre, dato che ciascuna finestra può a sua volta essere suddivisa a piacere, con il solo problema che se si creano troppe finestre non si vedranno altro che barre di stato.

All'interno di una finestra ci si può spostare con le frecce e gli altri tasti di spostamento e scrivere del testo, questo verrà sempre inserito nella posizione in cui si trova il cursore. Inoltre `emacs` prevede una grande quantità di combinazioni di tasti per andare avanti ed indietro, spesso tutt'altro che intuitive per chi proviene da altri sistemi, questo è una conseguenza del fatto che il programma è nato quando ancora le tastiere non avevano né frecce né altri tasti di funzione. In genere il testo viene inserito spostando quanto già presente nel file, a meno di non porsi nella modalità di sovrascrittura premendo il tasto INS (verrà notificato nella barra di stato) da cui si esce ripremendo lo stesso tasto.

I comandi invece vengono dati con delle combinazioni che prevedono la pressione contemporanea di un *tasto modificatore* (si chiamano così i tasti che hanno un effetto solo quando sono premuti in contemporanea ad altri tasti, come il tasto per la maiuscole, il *control*, ecc.) e di una lettera. Molti comandi inoltre prevedono l'uso di due combinazioni di tasti eseguite in successione. Data la complessità e la grande quantità di comandi esistono due modificatori per i comandi, il primo è il classico *control* ed il secondo è *alt* il cui uso può essere simulato, per le tastiere che non ce l'hanno, premendo prima il tasto di *escape* (ESC), e poi la lettera relativa.

Come già visto in sez. 2.1.2, dove la si è usata senza formalizzarne la definizione, per indicare la pressione contemporanea di un modificatore con un tasto qualunque "x" utilizzeremo una

notazione del tipo C-x per l'uso di *control* e M-x per l'uso di *alt*. L'uso di M- deriva dal fatto che il secondo tasto modificatore viene chiamato "*meta*" in tutta la documentazione di emacs, perché questo era il nome che aveva quando il programma è stato creato.

Operazione	**Combinazione di tasti**
aprire un file	C-x C-f seguito dal nome del file nel minibuffer (supporta il completamento automatico).
salvare un file	C-x C-s.
salvare con nome	C-x C-w seguito dal nome del file nel minibuffer (supporta il completamento automatico).
uscire	C-x C-c, se ci sono modifiche chiede se salvarle, scartarle o cancellare l'operazione.
annullare	C-_, C-/ o C-x u, ripetendo si torna ulteriormente indietro nell'annullamento.
seleziona	C-spazio marca l'inizio di una regione e si porta in modalità selezione, poi basta posizionarsi nel punto finale.
taglia	C-w una volta selezionata una regione.
incolla	C-y.
cancella	C-d in avanti e il tasto di backspace all'indietro.
ricerca	C-s seguito dal testo nel minibuffer esegue un ricerca incrementale sul testo specificato, C-s cerca il successivo, C-r cerca il precedente.
help	C-h ? poi scegliere nella finestra quello che si vuole.
annulla	(un comando) C-g o tre volte ESC.

Tabella 2.32: I principali comandi di emacs.

In tab. 2.32 si è riportata la lista delle principali operazioni e dei relativi comandi per eseguirle. I comandi elencati in tab. 2.32 sono attivi in qualunque modalità si usi emacs, infatti essendo un editor programmabile emacs può essere usato in modalità diverse a seconda del tipo di file su cui si sta lavorando, prevedendo in ciascun caso diverse serie di combinazioni di tasti e diversi comandi di manipolazione.

Per questo il programma ne mette però a disposizione molti altri sia generici che specifici per il tipo di file su cui si sta lavorando. Ad esempio scrivendo questo testo con LaTeX si hanno tutta una serie di comandi per mettere le parole negli indici, creare le intestazioni delle sezioni o delle tabelle, creare riferimenti a tabelle e figure presenti, ecc.

In generale poi la combinazione M-x consente di invocare qualunque funzionalità chiamando per nome la relativa funzione che la realizza. Dato che l'editor è programmabile anche funzionalità elementari sono realizzate tramite funzioni, ciascuna delle quali ha un nome, per cui se si vuole eseguire una correzione ortografica del testo corrente si potrà ad esempio invocare M-x ispell-buffer. Molti di questi comandi, ad esempio tutti quelli che richiedono l'immissione di un nome di file, usano il minibuffer per ricevere i relativi dati. Il minibuffer viene poi anche utilizzato per stampare messaggi e avvertimenti.

Inoltre in certi casi (come quello della specificazione di un file) la selezione supporta il meccanismo dell'autocompletamento (usando il tasto di tabulazione), e la capacità, qualora i completamenti siano multipli, di creare automaticamente una lista di selezione in una nuova finestra. In questo caso per spostarsi da una finestra all'altra occorrerà usare il comando C-x o (che ripetuto ritorna alla precedente), mentre per eliminare tutte le finestre presenti tranne quella dove è posizionato il cursore si potrà usare il comando C-x 1.

2.3.3 Un editor di base, `vi`

Uno degli editor più popolari, presente fin dagli inizi in tutti i sistemi Unix, è `vi`. Purtroppo i perché di questo nome si sono persi nei meandri del tempo, forse erano due lettere rimaste libere scelte a caso ... Questo editor deriva dagli editor di linea (si chiamano così i primi editor, nati ai tempi delle telescriventi, in cui si poteva modificare solo una riga alla volta) e ne eredita alcune caratteristiche. In particolare `vi` è un editor *modale*, in cui cioè sia i comandi che l'effetto di quanto si scrive dipendono dalla *modalità di operazioni* in cui il programma si trova al momento.

Questa caratteristica lo rende senz'altro assai poco intuitivo e abbastanza difficile da usare per il novizio. Ma i fan(atici) tendono invece a considerarla una caratteristica utile in quanto (secondo loro) con l'abitudine renderebbe più veloce le operazioni. Succede spesso però che al primo impatto non si riesca neanche ad uscire dall'editor, specie se capita di avere a che fare con una installazione che non ha attivato l'uso delle frecce.

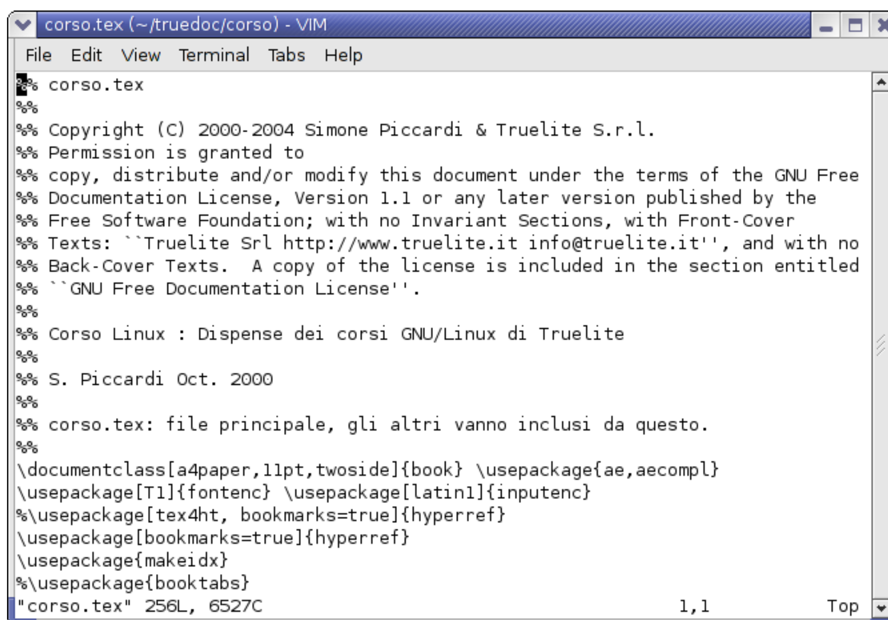


Figura 2.2: Schermata di avvio dell'editor `vi`.

Al contrario di `emacs`, di cui è il principale concorrente, `vi` si usa soltanto per manipolare file di testo, e pertanto non fornisce direttamente nessuna delle funzionalità più evolute di `emacs` (come la possibilità di fare debug dei programmi all'interno dell'editor facendo riferimento diretto al codice su cui si sta lavorando) ma resta comunque un editor molto potente.

Il principale vantaggio di `vi` è che essendo molto leggero e diffuso fin dalle prime versioni di Unix lo si trova installato praticamente su qualunque sistema e molto spesso è anche l'editor di default. Inoltre anche se le funzionalità del programma originale sono veramente minime, esistono alcune versioni più moderne, come `vim`, che hanno introdotto una lunga serie di capacità avanzate, come l'evidenziazione della sintassi. Non è detto però che quest'ultimo sia sempre

disponibile al posto del `vi` di base e non è detto che lo sia in una distribuzione di recupero, dato che con le funzionalità sono aumentate di pari passo anche le dimensioni.

Al solito `vi` si invoca passando come argomento il nome del file da modificare, il programma apre il file in una finestra unica (mostrata in fig. 2.2), dove compare il testo ed in cui ci si muove con le frecce,[30] lasciando libera l'ultima linea, usata per specificare i parametri di alcuni comandi o per ricevere le informazioni stampate dal programma, che nel caso sono il nome del file, la sua lunghezza in righe e caratteri e la posizione del cursore sulla prima colonna della prima riga.

Tutti le operazioni di `vi` sono eseguite con pressioni di singoli tasti, ma la possibilità di dare dei comandi al programma dipende dalla modalità in cui ci si trova. Come accennato infatti `vi` è un editor *modale* e supporta due modalità principali: la modalità *comando* e la modalità *inserimento*, ne esistono poi altre, come la modalità *sostituzione* o la modalità *selezione* di `vim`, ma non staremo ad entrare nei dettagli. In *modalità inserimento* i tasti premuti vengono utilizzati per scrivere il corrispondente carattere, in *modalità comando* invece ciascun tasto viene interpretato ed associato all'esecuzione di una specifica operazione.

Quando si avvia il programma questo si pone sempre in *modalità comando*, e si potranno eseguire le varie operazioni con la pressione del tasto corrispondente, in questo caso sullo schermo non sarà visualizzato nulla, se non l'effetto del comando eseguito. Tutti i comandi più complessi, ad esempio quelli che richiedono la scrittura di una espressione di ricerca o di un nome di file, devono essere dati dalla riga di comando dell'editor, che si attiva premendo "`:`", in questo caso il cursore si sposta dal testo e si pone sulla riga finale, dove si potrà inserire il resto dell'espressione.

Operazione	**Combinazione di tasti**
aprire un file	`:e` *`file`*
salvare un file	`:w`
salvare con nome	`:w` *`nomefile`*
uscire	`:q`, ma se ci sono modifiche non esce; nel caso `:wq` le salva mentre `:q!` le scarta
annullare	`u` solo sull'ultima modifica
seleziona	`v`, e poi ci si sposta con i tasti di freccia, ma è una estensione esclusiva di `vim`
taglia	`d`, ma vale con questo significato solo con una selezione fatta con `vim`
incolla	`p` inserisce l'ultimo oggetto cancellato.
cancella	`d` seguito da una altra lettera che specifica cosa cancellare; `x` cancella un carattere
ricerca	`/` seguito da un testo o una espressione regolare
annulla	ESC annulla un comando

Tabella 2.33: Come svolgere le principali operazioni con `vi`.

In tab. 2.33 si è riportata una lista delle principali operazioni relative ai compiti più comuni, associate ai rispettivi comandi. Si tenga presente però che questi possono essere attivati solo in modalità comando, se ci si è posti in modalità inserimento (o sostituzione) prima bisognerà uscirne con ESC.

La pressione dei tasti corrispondenti ai relativi comandi consente di cambiare modalità; ad esempio premendo "`i`" si passa in modalità inserimento, e si può iniziare a scrivere in corrispon-

[30] questo in teoria non è scontato per le versioni più elementari, ma quelle installate in tutte le distribuzioni di GNU/Linux se non altro supportano nativamente l'uso delle frecce.

Comando	Significato
h	spostamento a sinistra di un carattere.
j	spostamento in basso di una riga.
k	spostamento in alto di una riga.
l	spostamento a destra di un carattere.
w	spostamento in avanti di una parola.
b	spostamento indietro di una parola.
0	spostamento ad inizio riga.
$	spostamento a fine riga.
f	spostamento prima della prima occorrenza del carattere seguente.
t	spostamento dopo la prima occorrenza del carattere seguente.
i	va in modalità inserimento alla posizione corrente.
a	va in modalità inserimento al carattere seguente.
A	va in modalità inserimento a fine riga.
I	va in modalità inserimento a inizio riga.
o	inserisci una riga sotto quella corrente.
O	inserisci una riga sopra quella corrente.
x	cancella il carattere alla posizione corrente.
X	cancella il carattere precedente la posizione corrente.
s	cancella il carattere corrente e vai in inserimento.
S	cancella la riga corrente e vai in inserimento.
d*X*	seguito da un qualunque comando di spostamento *X* cancella dalla posizione corrente a quella dello spostamento.
dd	cancella la riga corrente.
D	cancella fino a fine riga.
c*X*	come d*X* ma si pone in inserimento eseguita la cancellazione.
cc	cancella la riga corrente e si pone in inserimento.
C	cancella fino a fine riga e si pone in inserimento.
r	rimpiazza il carattere alla posizione corrente.
R	sovrascrivi a partire dalla posizione corrente.
y*X*	seguito da un qualunque comando di spostamento *X* copia dalla posizione corrente a quella dello spostamento.
yy o Y	copia l'intera riga corrente.
p	incolla quanto cancellato dopo la posizione corrente.
P	incolla quanto cancellato prima della posizione corrente.
/*regex*	cerca l'espressione regolare *regex* in avanti.
?*regex*	cerca l'espressione regolare *regex* indietro.
n	cerca la corrispondenza successiva.
N	cerca la corrispondenza precedente.
v	entra in modalità *selezione* (*visual*) per effettuare una selezione di testo (solo su vim).
d	se dato in modalità *selezione* cancella quanto selezionato (solo su vim).
y	se dato in modalità *selezione* copia quanto selezionato (solo su vim).
.	ripete l'ultimo comando.
u	annulla l'ultimo comando (con vim può essere ripetuto).

Tabella 2.34: I principali tasti di comando diretti di vi.

denza del cursore. Ma ci sono altri comandi che permettono di entrare in modalità inserimento, come "a" che vi entra ma portandosi al carattere seguente o "A" che invece va a fine riga ed anche "o" che lo fa andando a capo creando una nuova riga. Se ne è riportato un sommario nella seconda sezione di tab. 2.34.

È allora abbastanza frequente che un utente alle prime armi, una volta aperto un file con vi,

tenti di scriverci e non veda nulla finché, premuto uno di questi tasti, non si porta in modalità inserimento. Il problema in genere è che una volta entrati in modalità inserimento l'unico modo per uscirne è quello di premere il tasto di *escape* (ESC); non sapendolo ci si troverà bloccati ed incapaci di uscire dal programma. In questo caso le versioni più recenti di `vim` sono più amichevoli, e reagiscono anche ad altre combinazioni di tasti, come `C-c`, dando delle indicazioni, ma con le versioni più elementari di `vi` non si ha nessun avviso e si resta completamente bloccati.

Una modalità simile a quella di inserimento è quella di *sostituzione*, in cui si entra premendo "`R`", in questo modo di nuovo tutti tasti assumeranno il loro valore letterale, ma quanto si scrive, invece di essere inserito in corrispondenza al cursore, spostando il testo già presente, sovrascriverà quest'ultimo. Anche da questa modalità si può uscire premendo il tasto di *escape* (ESC).

Come accennato `vi` supporta, essendo nato in un epoca in cui le tastiere non avevano i tasti freccia, lo spostamento nel file con i tasti "`h`", "`j`", "`k`" e "`l`" che effettuano rispettivamente lo spostamento a sinistra, basso, alto e destra. Un'altra caratteristica dell'editor è che qualunque comando dato in modalità comando può essere moltiplicato se lo si fa precedere da un numero, per cui per abbassarsi di 10 righe si potrà scrivere qualcosa tipo `10j`. Nella prima sezione di tab. 2.34 si è riportato un sommario dei principali tasti usati come comandi di spostamento.

Si può poi effettuare una ricerca sul testo con l'uso del tasto "`/`" per le ricerche in avanti e con "`?`" per le ricerche indietro, esattamente come per `less` e `more` che in effetti riprendono la sintassi da `vi`. Come per i due comandi citati la ricerca viene eseguita considerando la stringa da cercare una espressione regolare, diventa così possibile eseguire anche ricerche molto complesse.

Una delle caratteristiche che può lasciare più interdetti con `vi` è che non esiste il classico *taglia e incolla* in cui si seleziona una sezione qualunque del file, a meno che non usiate `vim`, che implementa questa funzionalità definendo una modalità *selezione* (o *visual*) in cui si entra con il comando "`v`". È possibile però cancellare caratteri singoli (con "`x`" per quello corrente, con "`X`" il precedente), parole (con `dw`) e righe (con `dd`), ed usare i moltiplicatori appena illustrati per cancellazioni multiple.

In generale il concetto è quello di far seguire a "`d`" uno dei comandi di spostamento (quelli della prima parte di tab. 2.34), e verrà effettuata la cancellazione fino a quel punto. Il testo cancellato viene memorizzato e lo si può poi incollare (anche più volte se si usano i moltiplicatori) con il tasto "`p`". Si sono riassunti i vari comandi di cancellazione nella terza sezione di tab. 2.34.

L'apertura di nuovi file deve essere fatta sulla riga di comando, e si esegue con `:e`, altri comandi che si possono utilizzare sono `:w` che salva il file e `:x` che salva ed esce, ed è equivalente al normale comando di uscita con salvataggio che è `:wq`; l'uscita scartando le modifiche invece si può forzare con `:q!`, per questi due comandi esistono anche delle versioni da dare in modalità normale e cioè rispettivamente `ZZ` e `ZQ`.

Un'altra caratteristica interessante della linea di comando di `vi` è che è possibile dare dei comandi di shell senza uscire dall'editor; una volta entrati sulla linea di comando del programma infatti si può usare il carattere "`!`" come *shell escape*, si potrà cioè usare una sintassi del tipo "`:!ls`" per ottenere la lista dei file su una schermata di servizio, da cui si potrà uscire, tornando alla modifica del file, premendo invio.

Un'altra funzionalità presente sulla linea di comando di `vi` è quella di sostituzione, attivabile con `:s`; questa funzionalità ha la stessa sintassi delle espressioni di sostituzione usate da `sed`, potremo cioè effettuare la sostituzione della stringa "*ric*" con la stringa "*sost*" utilizzando il comando "`:s/ric/sost/g`", ma come per le ricerche, potremo anche usare le espressioni regolari, ottenendo tutte le funzionalità che si sono descritte per `sed` in sez. 2.2.5.

Comando	**Significato**
`:h`	apre l'*help on line*.
`:N`	passa al file successivo.
`:q`	esce (se non ci sono modifiche).
`:q!` o `ZQ`	esce scartando le modifiche.
`:x` o `ZZ` o `:wq`	esce scrivendo le modifiche.
`:e` *`file`*	carica il file *`file`*.
`:e`	rilegge il file da disco.
`:e!`	rilegge il file da disco, scartando eventuali modifiche.
`:r` *`file`*	include il file *`file`* alla posizione corrente.
`:w` *`file`*	scrive sul file *`file`*.
`:w!`	salva il file anche se in modalità *readonly* (si deve essere proprietari del file).
`:!` *`cmd`*	esegue il comando *`cmd`*.
`:s/`*`ric`*`/`*`sub`*`/`	esegue la sostituzione, con la stessa sintassi di `sed` (vedi sez. 2.2.5).

Tabella 2.35: Le principali operazioni nella riga di comando di `vi`.

Infine un elenco dei principali comandi può essere ottenuto con la documentazione interna, accessibile con "`:h`". Si è riportato in tab. 2.35 l'elenco dei principali comandi che richiedono l'uso della riga di comando di `vi` (quelli che iniziano con "`:`") mentre la tabella riassuntiva di quelli diretti è in tab. 2.34, che è stata suddivisa in diverse sezioni relative ai comandi delle varie tipologie di operazioni.

2.3.4 Altri editor

Chiudiamo questa sezione con una breve panoramica sugli altri principali editor disponibili. Il primo di questi, un editor leggero e potente, è `joe`, gli affezionati del DOS noteranno che usa la sintassi di *wordstar*, un word processor testuale di quell'epoca. Offre tutte le normali funzionalità di un editor di testi e ha un help in linea per i vari comandi che si attiva con `C-k h`, che lo rende piuttosto facile da usare. Inoltre l'editor si può usare in modalità di emulazione usando le sintassi di altri editor; ad esempio invocandolo come `jmacs` userà la sintassi di `emacs`.

Al solito `joe` si invoca passando come argomento un nome di file, che viene aperto e visualizzato nella finestra principale, mostrata in fig. 2.3. La finestra presenta una riga di stato in testa, contenente i soliti dati (file, posizione, lunghezza ed il suggerimento per ottenere la schermata di aiuto), seguita dalla sezione principale in cui compare il testo del file ed in cui ci si muove con le frecce, l'ultima linea viene usata per mostrare i messaggi e per dare i comandi o ricevere le informazioni, e scompare quando non è necessaria.

In tab. 2.36 sono riportati i principali comandi per le funzionalità di base. Anche in questo caso l'editor non supporta il concetto classico del *taglia e incolla* in cui si rimuove il testo col taglia e poi lo si incolla altrove, ed usa invece l'approccio dello *sposta o copia* in cui prima si seleziona una sezione di testo e poi si specifica se spostarla oppure copiarla altrove.

Un altro editor per la *console* è `jed`, scritto come reimplementazione di `emacs` in C, che è molto più leggero di quest'ultimo (ma anche molto meno potente). Supporta anch'esso però un linguaggio di programmazione interno, ed è pertanto in grado di fornire parecchie funzionalità avanzate, e utilizzare diverse sintassi per i comandi a seconda del tipo di file. Fornisce anche

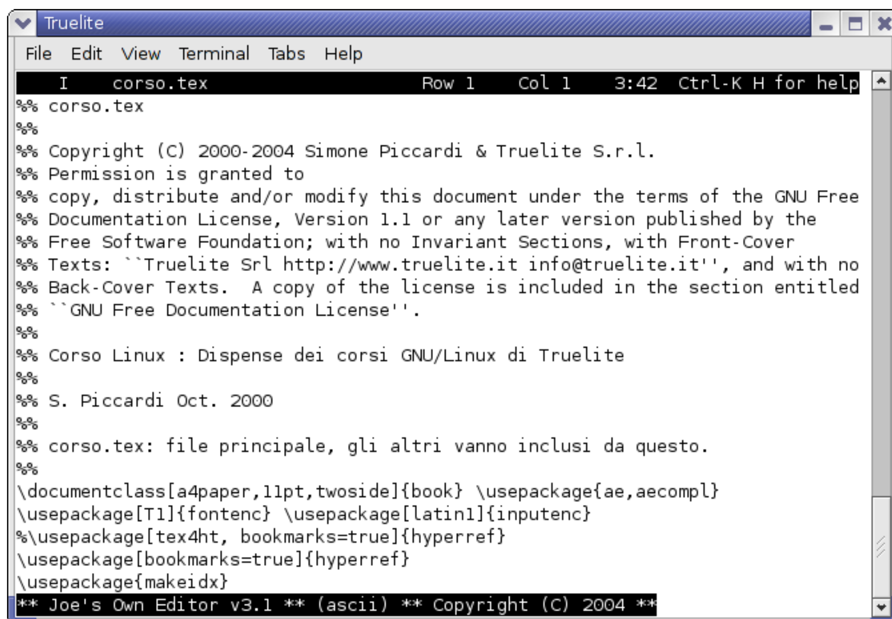


Figura 2.3: Schermata di avvio dell'editor `joe`.

Operazione	**Combinazione di tasti**
aprire un file	`C-k e` seguito dal nome del file.
salvare un file	`C-k x`.
salvare con nome	`C-k d` seguito dal nome del file.
uscire	`C-k x` esce e salva, `C-c` esce e se ci sono modifiche chiede d eventualmente scarta.
annullare	`C-_`, mentre con `C-^` si ripete l'ultima operazione.
seleziona	`C-k b` all'inizio e poi spostarsi con le frecce e dare `C-k k` alla fine.
sposta	`C-k m` muove la regione selezionata sulla posizione attuale.
copia	`C-k c` copia la regione selezionata sulla posizione corrente.
cancella	`C-k y` in avanti e *backspace* indietro.
ricerca	`C-k f` seguito da un testo esegue la prima ricerca, `C-L` cerca l'occorrenza successiva.
annulla	(un comando) `C-c`.

Tabella 2.36: I principali comandi di `joe`.

un'ampia capacità di evidenziazione della sintassi in *console* che con `emacs` è divenuta disponibile solo a partire dalla versione 21.

Essendo `jed` in sostanza un clone di `emacs` non staremo a ripeterne i comandi principali, che sono gli stessi di tab. 2.32, anche la composizione della finestra (una cui immagine è mostrata in fig. 2.4) è analoga a quella di `emacs` per cui basta rifarsi a quanto detto in precedenza; il vantaggio di `jed` è che l'utilizzo del menu è molto più intuitivo e avviene direttamente in modalità semigrafica anziché all'interno di finestre secondarie.

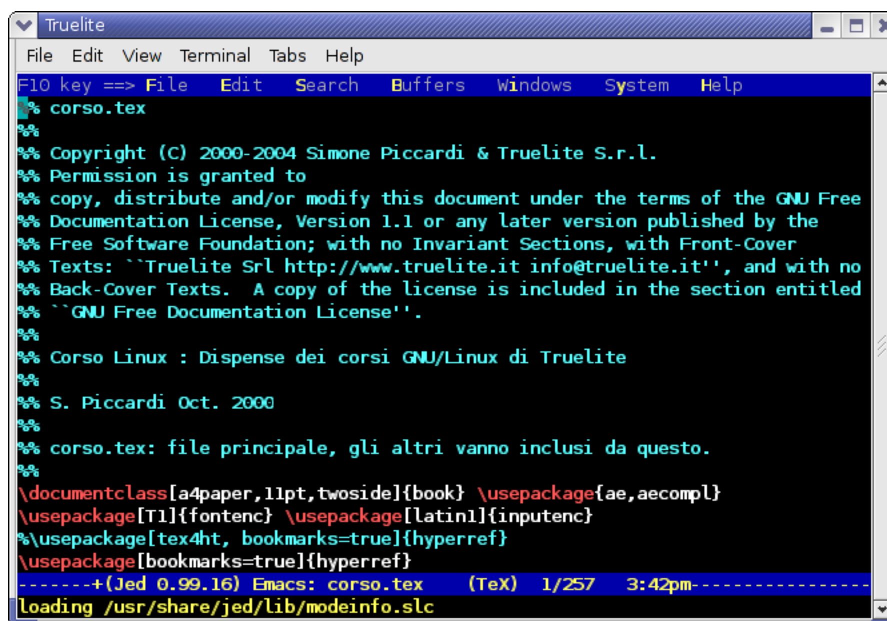


Figura 2.4: Schermata di avvio dell'editor `jed`.

Un altro editor abbastanza diffuso è `pico`; questo è derivato da `pine`, un vecchio client per leggere la posta elettronica in *console*, separando da esso l'editor interno usato per scrivere le email. Data la maggiore utilizzabilità rispetto a `vi` esso si è diffuso parecchio, il problema è che siccome `pine` non è software libero, non lo è neanche `pico`. Per questo motivo è stato realizzato un clone completamente libero chiamato `nano` identico dal punto di vista dei comandi principali, ma molto più leggero, anche più di `vi`, tanto da essere usato nei dischi di avvio di Debian per la shell di emergenza, e con qualche capacità in più.

Il programma si invoca al solito passando come argomento il nome del file da modificare, che viene aperto nella finestra mostrata in fig. 2.5, in cui si ha una riga di stato all'inizio, e due righe di aiuto in basso che riportano i comandi disponibili al momento, sulle quali vengono fatte pure le eventuali richieste di immissione o richieste scelte sulle azioni da compiere.

Un altro editor testuale che ha conosciuto un certo successo è `mcedit`. Anche questo è un sottoinsieme di un altro programma, `mc` (nome che sta per *Midnight Commander*), un file manager semigrafico utilizzabile in *console* che nasce come clone del *Norton Commander*. Anche questo è un editor che mantiene nella prima riga un riassunto dei comandi principali, per cui non staremo a descriverne i dettagli.

Tutti gli editor citati finora sono in grado di funzionare in modalità testuale su un terminale o dalla *console*, per completezza è opportuno citare che esistono tutta una serie di editor *grafici* come quelli inseriti in *Gnome* (`gedit`) e *KDE* (`kate`) che non sono spiegati qui in quanto l'interfaccia grafica è in grado di dare accesso alle funzionalità base con i soliti menu.

Fra gli editor grafici vale la pena segnalare uno dei più evoluti: `nedit` che dispone di un linguaggio di programmazione interno che ne permette una grande espandibilità ed è dotato di

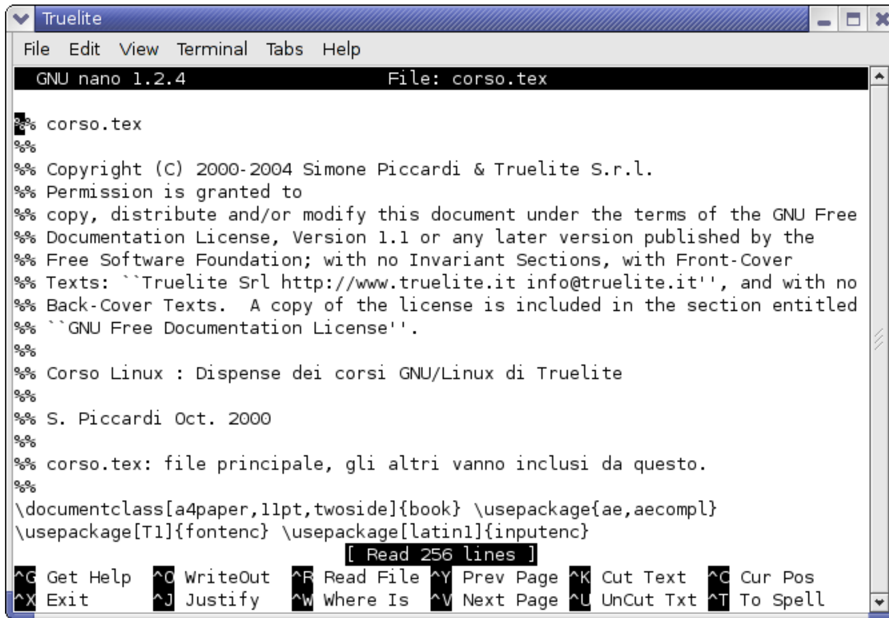


Figura 2.5: Schermata di avvio dell'editor `nano`.

moltissime funzionalità (forse il più vicino ad `emacs`); per questo è uno degli editor più potenti, pur restando anche facile da usare. Esistono infine anche versioni grafiche di alcuni degli editor precedenti (come `gvim` per `vi`), mentre come già accennato sia `emacs` che `xemacs` usati dall'interfaccia grafica mettono a disposizione menu e utilizzo del mouse.

2.4 Altri comandi di utilità

Dopo aver trattato i comandi che operano sui file, in questa sezione faremo una panoramica sugli altri principali comandi disponibili, attinenti funzionalità del sistema che finora non sono state prese in considerazione, come i comandi per eseguire manipolazioni avanzate sulla redirezione, quelli per la documentazione, quelli per gestire orologio e tempi di sistema, quelli per visualizzare le informazioni relative agli utenti ed al sistema e quelli che consentono di ottenere una serie di altre informazioni di vario tipo ad uso prevalentemente diagnostico.

2.4.1 I comandi di ausilio per la redirezione

Nonostante la grande flessibilità degli operatori di redirezione della shell, esistono situazioni in cui il loro uso non è sufficiente a fornire le funzionalità necessarie, per questo esistono dei comandi che permettono di estendere l'uso della redirezione.

Ad esempio uno dei fraintendimenti più comuni riguardo l'uso della concatenazione dei comandi è quello in cui si pensa di usare la *pipe* per passare come *argomenti* ad un comando successivo l'output di un comando precedente. Questo non è ovviamente possibile perché l'uso

di una *pipe* consente solo di passare lo *standard output* di un comando sullo *standard input* del successivo, e non ha nulla a che fare con gli argomenti di quest'ultimo.

Si ricordi infatti che gli argomenti provengono dall'analisi che fa la shell dalla linea di comando, spezzandola in parole, e che vengono passati in fase di messa in esecuzione di un programma, operazione che non ha nulla a che fare con la lettura, che avviene sempre a cura del comando stesso, dei dati presenti sullo *standard input*.

È però possibile fornire questa funzionalità con l'uso del comando `xargs`, il cui compito è replicare il comportamento della shell, che legge la linea di comando e ne estrae gli argomenti per poi passarli al comando che invoca. Il comando cioè effettua la lettura dello *standard input* e ne ricava una lista di argomenti da passare ad un successivo comando, in questo modo diventa possibile ricevere gli argomenti dall'output di un comando precedente tramite una *pipe*.

Il comando `xargs` prende come primo argomento il nome del comando da eseguire e come ulteriori argomenti le eventuali opzioni o argomenti iniziali da passare allo stesso. Il comando viene poi eseguito passandogli una lista di ulteriori argomenti, ottenuta dalla scansione dello *standard input* utilizzando gli spazi e gli a capo come separatori. È possibile proteggere la presenza di spazi all'interno degli argomenti come si fa con la shell, usando le virgolette o la barra trasversa. Se non si specifica nessun comando da eseguire, di default viene usato `echo`.

In questo modo se si ha una lista di file (o si è in grado di generarla) diventa possibile applicare a ciascun file della lista un comando qualunque usando semplicemente `cat` e `xargs`. Se cioè si vogliono cancellare tutti i file contenuti nel file `lista`, si potrà usare un comando del tipo:

```
cat lista | xargs rm -f
```

se invece li si vogliono copiare in una directory si potrà usare:

```
cat lista | xargs cp -t dir
```

ovviamente nel far questo occorrerà stare molto attenti, è sempre consigliabile infatti, quando si usa `xargs`, lanciare il comando senza argomenti per verificare che non si stiano facendo degli errori.

Il comando permette di impostare, con l'opzione `-E` una stringa da usare come marcatore per la fine del file, in modo da ignorare tutto quanto sarà ottenuto dopo. Con l'opzione `-t` permette inoltre di stampare a video il comando che verrà eseguito, e con `-p` di richiedere conferma dell'esecuzione sul terminale. Infine l'opzione `-0` richiede che quanto letto sullo *standard input* sia interpretato come stringhe terminate da uno zero (il carattere NUL), così da poter interpretare letteralmente i vari caratteri, in modo da non dover trattare in maniera speciale spazi, virgolette, ecc.

Questo consente di risolvere i problemi che si hanno quando si deve operare con `find` su dei file che hanno degli spazi o altri caratteri interpretati dalla shell nel nome; combinando l'opzione `-0` di `xargs` con l'opzione `-print0` di `find` si potranno trattare anche i nomi più complicati.

Si tenga presente che i due esempi precedenti per l'uso di `xargs` funzionano regolarmente perché i comandi utilizzati accettano come argomenti una lista di file, quando però si vogliono fare delle operazioni più complesse (ad esempio rinominare dei file sulla base di una elaborazione del loro nome) ci si trova di fronte al problema che programmi come `mv` si aspettano di avere due argomenti e non una lista.

In questo caso ci viene incontro una delle caratteristiche più interessanti di `xargs`, che è quella di poter eseguire invocazioni multiple dello stesso comando. Con l'opzione `-n` infatti si

può indicare il numero massimo di argomenti (sempre presi dalla lista ottenuta dalla scansione dello *standard input*) da passare a ciascuna invocazione, se si ricevono più argomenti i successivi saranno utilizzati per un'altra invocazione.

Se invece gli argomenti già sono suddivisi per linee, si può usare l'opzione `-L` che consente di indicare quante linee dello *standard input* utilizzare per ciascuna invocazione del comando. Così se volessimo rinominare tutti i file `.tex` di questo testo aggiungendo un `_new` al rispettivo nome potremmo usare qualcosa del tipo:[31]

```
ls *.tex | sed -r -e 's/(.*)(.tex)/\1\2 \1_new\2/g' | xargs -L1 mv
```

Si sono riassunte le principali opzioni di `xargs` in tab. 2.37; l'elenco completo delle opzioni ed una descrizione dettagliata del comando si possono trovare come sempre consultando la pagina di manuale, accessibile con `man xargs`.

Opzione	**Significato**
`-a` *`file`*	indica di leggere gli argomenti dal file *`file`* anziché dallo *standard input*.
`-E` *`str`*	indica la stringa *`str`* da considerare come fine del file, così da scartare tutto quello che segue, sostituisce nello standard POSIX l'opzione `-e` (tutt'ora presente) che ha lo stesso effetto.
`-n` *`N`*	usa al massimo *`N`* argomenti per ciascuna invocazione del comando.
`-L` *`N`*	usa un massimo di *`N`* linee come argomenti per ciascuna invocazione del comando.
`-p`	richiede conferma sul terminale prima di eseguire un comando.
`-t`	aumenta la prolissità del comando.
`-r`	non esegue nessun comando se non si ha niente in ingresso.
`-0`	richiede la terminazione dei singoli argomenti con un carattere nullo al posto dei normali spazi e disabilita l'interpretazione dei caratteri speciali.

Tabella 2.37: Principali opzioni del comando `xargs`.

Un secondo comando di ausilio per la redirezione è `tee`. Il suo scopo principale è quello di permettere di salvare su un file un risultato intermedio all'interno di una catena di redirezioni. In tal caso infatti l'uscita di un comando viene inviata sulla *pipe* e letta dal comando successivo, perdendo così la possibilità di esaminarla direttamente.

L'idea che sta alla base di `tee`, indicata dal suo stesso nome, è quella di creare una sorta di percorso a "T" nel flusso di dati della catena di redirezioni, in cui questi da una parte proseguono verso il comando successivo e dall'altra vengono salvati su un file. Questo significa che la maniera più comune di utilizzare `tee` sarà qualcosa nella forma:

```
cmd1 | tee result | cmd2
```

che consente di scrivere il risultato di `cmd1` sul file `result` ed al contempo inviarlo in *pipe* a `cmd2` per una ulteriore elaborazione.

Alternativamente si può usare il comando in coda ad una catena di redirezioni per salvare il risultato ed allo stesso tempo visualizzarlo sullo schermo. L'uso di `tee` infatti permette di

[31] si lascia come esercizio l'interpretazione dell'espressione regolare usata per `sed`.

effettuare una sorta di redirezione in *background*, in cui quello che il comando legge dallo *standard input* viene scritto sia sul file passato come argomento che sullo *standard output*. Il comando, come nella normale redirezione dello *standard output*, è anche in grado di eseguire la scrittura del file specificato in *append*, se invocato con l'opzione `-a`. Al solito per tutti i dettagli di funzionamento e l'elenco completo delle opzioni si consulti la pagina di manuale.

Un ultimo comando usato come ausilio per la redirezione è `yes`, che ha il semplice compito (se invocato senza argomenti) di scrivere continuamente sullo *standard input* una serie di "`y`". Anche in questo caso l'utilità di fare un comando per un compito così specifico diventa evidente solo considerando la capacità della shell di concatenare i comandi, per cui si può usare `yes` per *pilotare* automaticamente lo *standard input* di un comando che richiede conferme (come potrebbe essere `rm -i`). Il comando non ha nessuna opzione specifica e prende come argomento opzionale una stringa, che se indicata viene scritta sullo *standard input* al posto di "`y`".

2.4.2 I comandi per la documentazione

Benché talvolta possa risultare difficile trovare informazioni aggiornate sulle funzionalità più esoteriche, una delle caratteristiche di un sistema GNU/Linux è quella di essere fornito di una quantità impressionante di documentazione, tanto che una delle risposte più frequenti alle domande di chiarimento è RTFM, sigla che sta, a seconda dell'umore del momento, per *Read The Fine Manual* o *Read The Fucking Manual*.

Come accennato in sez. 2.2.1 ciascun comando di norma supporta di suo una opzione `--help` (spesso abbreviata in `-h`) che permette di visualizzarne brevemente la sintassi. Dato che questa informazione è in genere solo uno stringato riassunto delle opzioni disponibili, la fonte primaria delle informazioni relative ai comandi è nelle *pagine di manuale*, fin qui abbondantemente citate, gestite con il comando `man`.

Tutti i comandi di sistema prevedono infatti una pagina di manuale cui si accede semplicemente con la sintassi `man` *`comando`*. In particolare poi gli sviluppatori di Debian hanno come impegno preciso quello di fornire per ogni pacchetto la relativa documentazione. Ma le pagine di manuale non fanno riferimento solo ai comandi, il sistema infatti ha origine fin dai primi Unix e prevede la documentazione di tutto il sistema.

Sezione	**Significato**
`(1)`	programmi eseguibili o comandi di shell.
`(2)`	*system call* (funzioni fornite dal kernel).
`(3)`	funzioni di libreria.
`(4)`	documentazione sui file di dispositivo.
`(5)`	formati dei file di configurazione.
`(6)`	giochi.
`(7)`	varie (convenzioni, informazioni generiche su argomenti).
`(8)`	comandi di amministrazione.

Tabella 2.38: Sezioni delle pagine di manuale.

Per questo le pagine di manuale sono divise in sezioni, il cui numero è quello che compare fra parentesi nella prima riga di ciascuna pagina dopo il nome del comando in maiuscolo. Ciascuna sezione contiene la documentazione relativa ad un certo argomento, secondo la classificazione riportata in tab. 2.38, in cui però si sono messe solo le sezioni "*classiche*". Ci possono infatti

essere altre sezioni specifiche installate insieme ad alcuni pacchetti, ad esempio quelle della documentazione del linguaggio *Perl*, che non prenderemo in considerazione.

Con `man` si richiama la pagina di manuale, dove in genere si trova una documentazione esaustiva e dettagliata della sintassi e delle opzioni di un comando o del formato e del significato delle direttive di un file di configurazione. Il comando supporta una serie di opzioni che ne modificano il comportamento (riassunte in tab. 2.39) ed altre che consentono di controllare la formattazione per inviare l'output su stampante, al solito tutti i dettagli si trovano nella sua pagina di manuale, alla quale, come per gli altri comandi, si accede con `man man`.

Opzione	**Significato**
`-a`	mostra tutte le pagine di manuale che corrispondono.
`-k` *str*	cerca le pagine di manuale che hanno la stringa *str* nella descrizione (equivalente ad `apropos`).
`-f` *cmd*	elenca le pagine corrispondenti a `cmd` (equivalente a `whatis`).
`-w`	mostra dove è la pagina di manuale, non il suo contenuto.

Tabella 2.39: Principali opzioni del comando `man`.

Si tenga presente che il comando `man` richiama la pagina relativa al nome che si è passato come argomento, cercando in sequenza nelle varie sezioni e restituendo la prima che trova. Si tenga presente che la sequenza non è detto sia quella di tab. 2.38, essa infatti si imposta nella configurazione del sistema di gestione delle pagine di manuale, come vedremo a breve.

Per questo se esistono più versioni della stessa pagina in sezioni diverse (come ad esempio per il comando `passwd` e per il file `/etc/passwd`) verrà mostrata solo la prima, a meno di non aver specificato l'opzione `-a`. Se si vuole accedere alla seconda si dovrà richiamarla esplicitamente indicando la sezione come primo argomento, con un qualcosa del tipo `man 5 passwd`.

Il sistema delle pagine di manuale permette comunque di verificare se esistono più pagine associate ad uno stesso nome con il comando `whatis`, che stampa l'elenco delle pagine ad esso corrispondenti, così ad esempio avremo:

```
piccardi@anarres:~$ whatis passwd
passwd (1)           - change user password
passwd (5)           - The password file
```

Un'altra funzionalità utile del sistema delle pagine di manuale è fornita dal comando `apropos` che permette di effettuare la ricerca della parola passata come argomento fra le descrizioni brevi dei comandi che compaiono nella intestazione delle pagine di manuale (quelle appena mostrate anche nell'output di `whatis`) per cui potremo eseguire la ricerca:

```
piccardi@anarres:~$ apropos "user password"
chage (1)            - change user password expiry information
passwd (1)           - change user password
```

e si noti come si siano usati i doppi apici per effettuare una ricerca su una stringa contenente uno spazio, dato che altrimenti si sarebbero passate due stringhe al comando.

Come accennato il comando `man` supporta la possibilità di generare documentazione stampabile. Questo avviene perché in realtà le pagine di manuale non sono scritte direttamente nel formato in cui le vediamo su video, ma in uno speciale linguaggio di formattazione, chiamato *troff*, che permette la creazione di sezioni, indici, ecc. Per questo il testo mostrato sul terminale, è solo uno fra i vari formati di uscita disponibili.

Tutte le volte che si richiama il comando `man` per visualizzare una nuova pagina di manuale il comando dovrà recuperare uno di questi file generando un testo in formato opportuno per la visualizzazione. Per evitare di ripetere il procedimento della generazione ad una successiva invocazione tutti questi file vengono salvati, usualmente sotto `/var/cache/man/`, in una directory `catN`, dove *N* corrisponde alla sezione cui appartiene la pagina, così da poterli recuperare velocemente.

Un secondo aspetto del sistema delle pagine di manuale è quello dell'indicizzazione dei contenuti usata dai comandi `whatis` e `apropos`; questa infatti non è diretta (come si può verificare tutte le volte che si installa un nuovo pacchetto e non lo si trova nei risultati di detti comandi), ma viene effettuata normalmente tramite il programma `mandb`, che costruisce gli opportuni file con gli indici usati dai precedenti.

In genere il programma viene invocato periodicamente nelle operazioni giornaliere eseguite da `cron` (vedi sez. 3.2.1), ma lo si può anche invocare direttamente; le opzioni principali sono `-c` che gli fa ricreare tutti gli indici da zero (il default è aggiornare quelli esistenti) e `-p` che non fa eseguire la ripulitura degli indici dei dati della pagine non più presenti; per la descrizione completa delle altre opzioni ed i dettagli sul comando al solito si faccia riferimento alla sua pagina di manuale.

Per stabilire in quale directory si trovano gli originali delle pagine, in che ordine cercarle, dove devono essere memorizzate le pagine riformattate per la visualizzazione, dove mantenere gli indici delle pagine presenti e delle parole su cui effettuare le ricerche, il sistema di gestione delle *man pages* usa il file di configurazione `/etc/manpath.config`.

Il primo aspetto della configurazione è quello della definizione delle directory in cui si trovano le pagine di manuale; come per gli eseguibili infatti anch'esse possono essere installate in diverse sezioni dell'albero. Si pensi infatti al caso in cui si siano installate diverse versioni di uno stesso pacchetto, ad esempio una versione più recente a mano sotto `/usr/local`, in tal caso le pagine di manuale saranno sotto `/usr/local/man` ed il sistema deve essere in grado di vederle, ed in un certo ordine rispetto alle altre. Una lista di tutte queste directory si può stampare con il comando `manpath`, ed in genere si otterrà qualcosa del tipo:

```
piccardi@anarres:~$ manpath
/usr/local/man:/usr/local/share/man:/usr/share/man
```

a meno di non aver definito in maniera diretta la variabile di ambiente `MANPATH` (nel qual caso sarà semplicemente stampato il suo valore) che permette di soprassedere il valore preimpostato dal sistema.

La configurazione delle directory in cui cercare le pagine di manuale è una di quelle impostazioni mantenute in `manpath.config`. La direttiva `MANDATORY_MANPATH` serve per indicare tutte le directory che si vuole siano sempre aggiunte (se esistono) alla lista di quelle in cui si cercano le pagine di manuale; la direttiva prende come argomento il pathname alla directory contenente gli originali, e deve essere ripetuta per tutte le directory che si vuole siano aggiunte alla lista.

La direttiva `MANPATH_MAP` indica invece la corrispondenza fra le directory dei comandi e le relative pagine di manuale; serve per aggiornare dinamicamente la lista delle directory in cui cercare le pagine di manuale sulla base delle directory presenti nel `PATH` (vedi sez. 2.1.3) dell'utente. La direttiva richiede due argomenti; il primo indica una directory contenente i comandi, il secondo la directory delle corrispondenti pagine di manuale. Se la prima compare nel `PATH` dell'utente la seconda è aggiunta al `MANPATH` in maniera implicita.

Infine la direttiva `MANDB_MAP` serve ad indicare a `mandb` dove devono essere messi i file degli indici, e dove devono essere create le pagine temporanee. Un estratto del file `manpath.config`, così come installato su una Debian Sid, è il seguente:

manpath.config
```
MANDATORY_MANPATH                     /usr/man
MANDATORY_MANPATH                     /usr/share/man
MANDATORY_MANPATH                     /usr/local/man
...
MANPATH_MAP      /bin                 /usr/share/man
MANPATH_MAP      /usr/bin             /usr/share/man
MANPATH_MAP      /sbin                /usr/share/man
MANPATH_MAP      /usr/sbin            /usr/share/man
MANPATH_MAP      /usr/local/bin       /usr/local/man
...
MANDB_MAP        /usr/man             /var/cache/man/fsstnd
MANDB_MAP        /usr/share/man       /var/cache/man
MANDB_MAP        /usr/local/man       /var/cache/man/oldlocal
MANDB_MAP        /usr/local/share/man /var/cache/man/local
MANDB_MAP        /usr/X11R6/man       /var/cache/man/X11R6
MANDB_MAP        /opt/man             /var/cache/man/opt
...
SECTION          1 n l 8 3 2 3posix 3pm 3perl 5 4 9 6 7
```

L'ultima direttiva, `SECTION`, indica l'ordine di ricerca delle pagine di manuale nelle varie sezioni, si noti come nell'esempio appena citato ci siano altre sezioni oltre a quelle classiche illustrate in tab. 2.38.

Oltre alla consultazione delle pagine di manuale, una seconda fonte di informazioni e documentazione è costituita dal sistema di *help on line* fornito dal comando `info` e dalle cosiddette *info pages*. In questo caso si tratta di una forma alternativa di strutturazione delle informazioni che usa un formato diverso e che aggiunge, rispetto alle pagine di manuale, delle caratteristiche più avanzate e molto comode come la possibilità di navigare usando link ad altre pagine, una organizzazione gerarchica strutturata ad albero con nodi e sezioni, la possibilità di consultare indici, ecc.[32] Grazie a questa struttura `info` permette anche di accedere a documenti più complessi di una semplice pagina di manuale, e vi si trovano una grande quantità di manuali di grandissima importanza, come documentazione sui sistemi di sviluppo, sui linguaggi, sulle librerie, ecc.

Date le sue caratteristiche evolute *info* è il formato raccomandato dal progetto GNU, che lo considera alternativo rispetto alle pagine di manuale. Molti però continuano ad utilizzare quest'ultime per cui alla fine i due sistemi si trovano a convivere. In genere il contenuto delle pagine di manuale è direttamente accessibile anche con il comando `info comando`, ma usualmente è disponibile anche un'informazione di livello più alto; ad esempio la documentazione dettagliata dei comandi più complessi della *Unix toolbox*, che comprende anche le spiegazioni generali sulla architettura scelta, viene mantenuta nelle pagine *info* del relativo pacchetto, accessibili con `info coreutils`.

Se ci si limita a richiamare il comando `info` ci verrà mostrata la radice del sistema dove sono elencati tutti i documenti installati, e sarà possibile iniziare la navigazione spostandosi con le

[32] tanto che queste informazioni vengono utilizzate anche da altri programmi, come i vari sistemi di *help on line* degli ambienti grafici.

frecce, seguire i link premendo invio, tornare al livello precedente premendo "u", passare al nodo successivo con "n", tornare al nodo precedente con "p", andare all'indice con "i", effettuare una ricerca con "/", ed infine visualizzare i vari comandi disponibili con "?". La potenza del sistema è che, come per le pagine di manuale, i testi vengono generalmente scritti in un apposito linguaggio di formattazione, per cui la versione testuale è solo una delle possibili forme di visualizzazione, e si può facilmente accedere alla stessa documentazione anche dall'interfaccia grafica con gli opportuni programmi di consultazione.

Si tenga presente poi che spesso i pacchetti dei singoli programmi vengono distribuiti con la documentazione prodotta direttamente dagli autori, che di norma, seguendo le indicazioni del *Filesystem Hierarchy Standard*, viene installata in una directory `/usr/share/doc/nomeprogramma`. A questa documentazione specifica poi si aggiunge il patrimonio degli HOWTO, una serie di documenti sul *come fare* a riguardo di un certo argomento, disponibili in diversi formati (dal testo puro, all'HTML, al PDF). Questi sono stati raccolti dal *Linux Documentation Project* (`http://www.tldp.org`), e di norma vengono installati dalle varie distribuzioni nella directory `/usr/share/doc/HOWTO/`.

Infine l'utilizzo dei motori di ricerca è un ottimo sistema per recuperare le informazioni pubblicate su Internet, ed alcuni di essi forniscono anche un accesso specifico per ricerche su questo argomento. Fra queste troverete le versioni pubblicate degli HOWTO (anche se queste vengono spesso distribuite come documentazione nel sistema, non è detto che siano complete o aggiornate) ma anche altre risorse come le annotazioni ed i log, le istruzioni lasciati in innumerevoli pagine web e gli archivi delle liste di discussione dei vari progetti in cui qualcuno può aver già avuto il vostro problema e trovato una risposta.

2.4.3 I comandi per la gestione dei tempi

Prima di accennare ai principali comandi per la gestione di tempo e date in GNU/Linux occorre una breve introduzione sulla gestione del tempo nei sistemi Unix. Storicamente i sistemi unix-like hanno sempre mantenuto due distinti tipi di tempo all'interno del sistema: essi sono rispettivamente chiamati *calendar time* e *process time*, secondo le seguenti definizioni:

calendar time o *tempo di calendario*. È il numero di secondi dalla mezzanotte del primo gennaio 1970, in *tempo universale coordinato* (o UTC), data che viene usualmente indicata con `00:00:00 Jan, 1 1970 (UTC)` e chiamata "*the Epoch*". Questo tempo viene spesso chiamato anche GMT (*Greenwich Mean Time*) dato che l'UTC corrisponde all'ora locale di Greenwich.[33]. È il tempo su cui viene mantenuto l'orologio del kernel, e viene usato ad esempio per indicare le date di modifica dei file o quelle di avvio dei processi.

process time o *tempo di processore*. È il tempo usato internamente dal kernel per le sue temporizzazioni. In genere è legato alle interruzioni del timer hardware, e viene impiegato per tutti i calcoli dello scheduler, è pertanto il tempo in cui viene misurato il tempo di esecuzione dei processi.

In genere il tempo a cui si fa riferimento è sempre il *calendar time*, il *process time* viene usato soltanto dai comandi che riportano le proprietà specifiche dei processi (come `ps` o `top`)

[33]in realtà con GMT si fa alle volte riferimento al tempo solare misurato dall'osservatorio di Greenwich, che è leggermente diverso, per i dettagli si consulti `http://en.wikipedia.org/wiki/UTC`

relative ai tempi di esecuzione degli stessi. Oltre a questi due, già trattati in sez. 1.3.2, l'unico altro comando comune che usa il *process time* è `time`, che prende come argomento una riga di comando da eseguire, e stampa a video, alla terminazione dell'esecuzione di quest'ultima, tre valori di tempo espressi in *process time*, che sono il tempo totale impiegato per l'esecuzione del programma (con la sigla `real`), il tempo passato nell'esecuzione di codice in user space (con la sigla `user`), ed il tempo passato nell'esecuzione di codice dentro il kernel, cioè all'interno delle *system call* invocate dal processo (con la sigla `sys`), ad esempio:

```
piccardi@hain:~$ time ls
CVS  README  corso  internet-server  ldap  lucidi  samba

real    0m0.030s
user    0m0.000s
sys     0m0.010s
```

e si noti come il tempo *reale* è sempre maggiore degli altri due in quanto tiene conto anche del tempo in cui il processo è stato in stato di *sleep* in attesa di I/O.

Il comando prende varie opzioni, le principali delle quali sono `-o` che permette di specificare un file su cui scrivere i risultati al posto dello *standard output*, e `-f` che permette di specificare un formato per i tempi. La descrizione completa del comando (comprese le stringhe usate per l'opzione `-f`) si trova al solito nella pagina di manuale.

Tutti gli altri comandi relativi ai tempi hanno a che fare con la gestione del *calendar time*. Qui si aggiungono ulteriori complicazioni; la prima è che esistono due orologi, quello di sistema, usato dal kernel per tutte le sue operazioni, ed aggiornato via software dal kernel stesso, e l'orologio hardware che funziona in maniera del tutto indipendente e che rimane in funzione anche quando la macchina è spenta.[34] Di solito il kernel legge l'orologio hardware all'avvio, per impostare il valore iniziale dell'orologio di sistema, e poi non lo considera più.

La seconda complicazione è che il kernel non conosce assolutamente i fusi orari; per lui il tempo è assoluto, e fa sempre riferimento al tempo standard universale (l'UTC appunto). Tutta la gestione dei fusi orari è demandata alle applicazioni in user space che, tutte le volte che leggono un tempo, devono essere in grado di convertirlo nell'ora locale, il tutto è fatto attraverso delle opportune funzioni di libreria, che permettono di eseguire il compito in maniera trasparente, torneremo sull'argomento con maggiori dettagli in sez. 3.1.4.

Se ci si pensa questa è la scelta più logica: non solo si evita di inserire una serie di informazioni complesse all'interno del kernel, ma facendo così i tempi di sistema sono sempre coerenti, e non dipendono dal fuso orario in cui ci si trova, nel senso che i file modificati un'ora fa risulteranno sempre con l'ora giusta qualunque cambiamento sia stato fatto nel fuso orario. Purtroppo altri sistemi operativi usano l'ora locale per l'orologio di sistema che coincide con l'orologio hardware, con tutti i problemi che questo comporta quando si deve cambiare fuso orario (con file che possono anche risultare essere stati creati nel futuro), e soprattutto di incompatibilità dei tempi in caso di dual boot.

Il principale comando che permette di leggere ed impostare l'orologio di sistema è `date`. Se usato senza argomenti questo si limita a stampare data ed ora corrente nel formato standard:

```
piccardi@hain:~$ date
Fri Sep 19 12:45:42 CEST 2003
```

[34] è l'orologio che si trova sulla scheda madre, ed è alimentato dalla batteria che si trova su di essa.

il comando prende come argomento o il tempo da impostare (operazione privilegiata che può eseguire solo l'amministratore) o un formato di stampa per il tempo che modifica l'output del comando facendogli stampare solo le parti di data volute.

Se si vuole impostare la data questa deve essere specificata con una stringa nella forma `MMDDhhmm[[CC]YY][.ss]`, dove i termini fra parentesi quadra sono opzionali. Con `MM` si indica il mese (in numero), con `DD` il giorno, con `hh` l'ora, con `mm` il minuto, mentre l'anno si indica o con le cifre finali `YY` o per intero con tanto di secoli come `CCYY`, infine i secondi `ss` devono essere preceduti da un punto; ad esempio si può provare ad impostare l'orologio con:

```
piccardi@hain:~$ date 09191854
date: cannot set date: Operation not permitted
Fri Sep 19 18:54:00 CEST 2003
```

e l'operazione fallisce, dato che non si è l'amministratore, ma il tempo che si voleva impostare viene comunque stampato a video.

Se invece l'argomento che si passa inizia con un "`+`" esso viene interpretato come una stringa di formattazione per il risultato del comando. La stringa usa il carattere "`%`" per indicare una direttiva di formattazione che verrà sostituita dall'opportuno valore, tutti gli altri caratteri resteranno immutati. Le principali direttive sono riportate in tab. 2.40, l'elenco completo è nella pagina di manuale.

Direttiva	**Significato**
`%%`	il carattere `%`.
`%a`	il nome del giorno della settimana abbreviato, secondo la localizzazione.
`%A`	il nome del giorno della settimana completo, secondo la localizzazione.
`%b`	il nome del mese abbreviato, secondo la localizzazione.
`%B`	il nome del mese completo, secondo la localizzazione.
`%c`	data e orario, secondo la localizzazione.
`%d`	giorno del mese, nella forma `(01..31)`.
`%D`	data, nella forma `(mm/dd/yy)`.
`%H`	ora del giorno, nella forma `( 0..23)`.
`%I`	ora del giorno, nella forma `( 1..12)`.
`%m`	mese, nella forma `(01..12)`.
`%M`	minuto, nella forma `(00..59)`.
`%r`	orario nella forma `(hh:mm:ss [AP]M)` su 12 ore.
`%T`	orario nella forma `(hh:mm:ss)` su 24 ore.
`%S`	numero di secondi.
`%w`	giorno della settimana, nella forma `(0..6)` a partire dalla domenica.
`%x`	data nella rappresentazione secondo la localizzazione.
`%y`	anno abbreviato alle ultime due cifre.
`%Y`	anno completo.
`%Z`	nome convenzionale del fuso orario.

Tabella 2.40: Direttive di formattazione per il comando `date`.

Il comando prende inoltre varie opzioni, le principali delle quali sono `-d` che permette di specificare una data da stampare al posto del tempo corrente, e `-s` che permette, con lo stesso formato, di specificare l'impostazione dell'orologio di sistema. L'utilità di queste due opzioni è che la stringa che prendono come parametro può essere nelle forme più varie, e non solo riconosce tutti i formati standard che si ottengono con le direttive di tab. 2.40, ma anche descrizioni verbali (solo in inglese, però) come `8 day ago`, o `1 month`. La descrizione di queste stringhe può essere

trovata solo nelle pagine info del comando, accessibili con `info date`, dove si trova anche la trattazione completa di tutte le funzionalità.

Il secondo comando che riguarda la gestione del tempo è `hwclock`, che permette di impostare l'orologio hardware. Il comando non prende argomenti, e lo si utilizza attraverso le opzioni. Se non si specifica nessuna opzione il comando si limita leggere il valore dell'orologio hardware e a stamparlo sullo *standard output*, con un risultato del tipo:

```
root@hain:~# hwclock
Tue Sep 23 15:36:58 2003  -0.763201 seconds
```

(risultato che si ottiene anche con l'opzione `-r` o `--show`).

Il comando poi permette di impostare l'orologio hardware con l'opzione `--set`, cui deve seguire un `--date` che specifichi la data da usare nel formato con cui la si specifica anche per `date`. Si può però usare l'opzione `-w` (o `--systohc`) per impostare l'ora direttamente al tempo segnato dall'orologio di sistema. Viceversa l'opzione `-s` (o `--hctosys`) imposta l'orologio di sistema al tempo di quello hardware.

Infine un altro comando molto utilizzato nella gestione dell'orologio di sistema è `ntpdate` che consente di sincronizzare l'ora con un server NTP (*Network Time Protocol*) esterno. In genere infatti l'orologio di sistema di una macchina non è molto preciso, e tende ad andare avanti o restare indietro a seconda delle condizioni di carico; anche l'orologio hardware, pur essendo fornito da un quarzo interno, risente delle variazioni di temperatura e di tensione sulla piastra madre. Per questo motivo si possono avere delle derive significative nei tempi.

Il servizio NTP è un servizio di rete (che tratteremo in sez. 8.1.4) a cui ci si può rivolgere per poter *rimettere* l'orologio di una macchina, il comando `ntpdate` effettua una richiesta ad un server NTP, il cui indirizzo viene passato come argomento,[35] e sincronizza l'orologio di sistema con l'orario fornito dal server. Il modo più comune per invocare il programma è:

```
root@hain:~# ntpdate pool.ntp.org
28 Dec 9:51:58 ntpdate[2199]: adjust time server 44.4.14.88 offset 0.006230 sec
```

Si può invocare il comando con l'opzione `-q` per fargli semplicemente eseguire una richiesta senza aggiornare l'ora, e l'opzione `-v` per fargli stampare più informazioni, le restanti opzioni, ed i dettagli relativi al funzionamento del programma, sono al solito riportati nella pagina di manuale.

Si tenga presente che se il programma rivela una differenza di tempo maggiore 0.5 secondi effettua una sincronizzazione immediata, e si avrà pertanto uno *scalino* nella scala dei tempi, mentre se la differenza è inferiore provvede ad accelerare (o decelerare) l'orologio di sistema finché la differenza non sia stata compensata.[36]

2.4.4 I comandi per le informazioni sugli utenti e il sistema

Tratteremo in questa sezione una serie di comandi utilizzabili per ottenere informazioni sulle identità degli utenti, i gruppi cui appartengono, gli accessi al sistema e altre caratteristiche

[35] diamo per scontata la conoscenza del significato di indirizzo, l'argomento viene affrontato nei dettagli in sez. 7.2.1 e sez. 7.4; il comando accetta sia indirizzi IP in forma numerica, che nomi a dominio in forma simbolica, come quello dell'esempio.

[36] i dettagli sulle funzioni con cui viene effettuato l'aggiornamento vanno al di là degli scopi di questo testo, si può trovare una trattazione dell'argomento in sez. 8.4.3 di [GaPiL].

generali del sistema. Un primo comando, che permette di verificare con quale utente ci si è collegati al sistema, è `whoami`, che stampa il valore del proprio *username* con qualcosa del tipo:

```
piccardi@anarres:~$ whoami
piccardi
```

e se può sembrare superfluo verificare con quale utente ci si è collegati (cosa che di norma ci viene ricordata anche dal prompt della shell) si pensi invece a quanto può essere utile nel caso di uno script di shell che debba eseguire compiti diversi in base all'utente che lo esegue.

Per verificare di quali gruppi fa parte un utente si può usare invece il comando `groups`. Questo, se invocato senza nessun argomento, stampa l'elenco dei nomi dei gruppi a cui si appartiene, ad esempio:

```
piccardi@anarres:~$ groups
piccardi voice cdrom audio video plugdev lpadmin fuse scanner saned vboxusers
```

se invece si specifica come argomento un nome utente, il comando stampa la lista di gruppi cui appartiene l'utente indicato. Sia `groups` che `whoami` non hanno opzioni se non quelle standard GNU viste in sez. 2.2.1.

Il comando più generale è però `id`, che consente di ottenere i valori di tutti gli identificativi associati ad un utente. Il comando stampa i valori di utente e relativi gruppi sia in forma letterale che numerica, e permette di scegliere se stampare quelli reali o quelli effettivi (si ricordi quanto visto in sez. 1.4.1); un esempio del suo utilizzo è:

```
piccardi@anarres:~$ id
uid=1002(piccardi) gid=1002(piccardi) groups=22(voice),24(cdrom),29(audio),44(vi
deo),46(plugdev),117(lpadmin),119(fuse),120(scanner),122(saned),124(vboxusers),1
002(piccardi)
```

in cui, invocato senza argomenti, stampa tutti gli identificativi associati all'utente corrente.[37]

Oltre alla possibilità di passare un argomento che specifica l'utente di cui si vogliono ottenere gli identificativi, il comando prende molte opzioni che permettono di indicare nei dettagli quali informazioni si vogliono stampare ed in che formato; con `-u` si può chiedere di mostrare solo l'*user ID*, con `-g` il *group ID*, è inoltre possibile chiedere la visualizzazione dei identificativi del gruppo *real* (quelli dell'utente che ha lanciato il processo) invece degli *effective* (quelli per conto di cui gira il processo, mostrati di default) con l'opzione `-r`. Al solito si può fare riferimento alla pagina di manuale, accessibile con `man id`, per la documentazione completa.

Oltre ai dati relativi agli identificativi associati ai singoli utenti, sulla cui gestione amministrativa torneremo in sez. 4.3, il sistema mantiene anche una serie di informazioni sulle loro attività, che possono essere richieste con altrettanti comandi. In particolare tutte le volte che un utente si collega al sistema e avvia un terminale l'evento viene registrato in uno specifico formato[38] nei file `/var/log/wtmp` e `/var/run/utmp` (il primo contiene le informazioni storiche, il secondo quelle correnti) insieme ad altre informazioni relative sia alla modalità in cui avviene l'accesso che ad altri eventi di sistema. In particolare vengono registrati in questi file tutti i riavvii ed in generale tutti i cambiamenti di *runlevel* (vedi sez. 5.3.7) che sono stati eseguiti.

[37] ad essere pignoli, si tratta in realtà degli identificativi associati al processo che esegue `id`, che prevedono sia l'*user ID* che i vari *group ID*, nel caso specifico quelli del gruppo *real*.

[38] per i dettagli si può consultare sez. 6.4.2 di [GaPiL].

I dati vengono registrati in questi file in forma binaria, ma possono essere consultati con l'ausilio di alcuni comandi dedicati, che consentono di ottenere informazioni sugli accessi degli utenti ed anche sulle attività svolte nel sistema. Ad esempio l'elenco degli utenti attualmente collegati si può ottenere con l'uso del comando `users` che stampa a video una semplice lista degli stessi, da usare ad esempio in uno script che debba inviare avvisi, con qualcosa del tipo:

```
piccardi@anarres:~$ users
root piccardi
```

Una lista più dettagliata può essere ottenuta con il comando `who`, che stampa l'elenco in forma di tabella, riportando per ciascun utente collegato al sistema il relativo *username* seguito dal terminale su cui questo è attivo, dall'ora in cui è avvenuto l'accesso e dall'eventuale macchina remota o dal *display* dell'interfaccia grafica (vedi sez. 3.3.4) da cui questo è avvenuto, cioè qualcosa del tipo:

```
piccardi@anarres:~$ who
piccardi tty1         2008-03-26 00:02
piccardi pts/0        2008-03-27 12:45 (192.168.0.3)
piccardi pts/1        2008-03-23 12:07 (:0.0)
```

che mostra tre sessioni dello stesso utente, una effettuata da una console locale, una da una macchina remota ed una dall'interfaccia grafica.

Il comando supporta molte opzioni (al solito per i dettagli si consulti la pagina di manuale), volte a stampare le varie informazioni disponibili, ma in generale lo si usa senza argomenti per avere la lista degli utenti come illustrato. Se gli si passa un argomento questo verrà considerato il nome del file da cui leggere le informazioni,[39] mentre con due argomenti stampa soltanto la voce relativa all'utente che lo ha lanciato.

In quest'ultimo caso si tende ad usarlo nella forma discorsiva `who am i`, che ricorda il precedente `whoami`, ma oltre al risultato, che presenta il nome dell'utente insieme al terminale e alle altre informazioni mostrate nel precedente esempio,[40] è diverso anche il significato, `whoami` infatti riporta il nome dell'utente associato all'*user ID* del processo corrente e dà sempre un risultato, mentre `who am i` fa riferimento all'utente effettivamente collegato, per cui se usato all'interno di uno script eseguito in modalità non interattiva (ad esempio lanciato con `cron`, vedi sez. 3.2.1) non darà nessun risultato.

Come alternativa a `who` esiste anche il comando `w`, che esegue lo stesso compito ma stampa delle informazioni più dettagliate. In particolare il comando stampa sempre una riga introduttiva, analoga a quella mostrata anche da `top`, che riporta ora, tempo trascorso dall'ultimo riavvio (l'*uptime*), numero di utenti connessi e carico medio della macchina, cui segue la lista degli utenti in cui, oltre alle informazioni già riportate da `who` ci sono pure i dati relativi al tempo di CPU usato dall'utente ed il comando corrente; si avrà cioè:

```
piccardi@anarres:~$ w
 14:49:36 up 4 days,  2:45,  3 users,  load average: 0.00, 0.01, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
piccardi tty1     -                Wed03   35:38   0.76s  0.73s -bash
```

[39] il comando usa `/var/run/utmp` che contiene le informazioni correnti, ma gli si può passare come argomento `/var/log/wtmp` per fargli leggere quelle archiviate.

[40] qualora si voglia stampare il nome utente è possibile usare il comando `logname`, ma anche questo fa riferimento all'utente collegato.

```
piccardi pts/0    192.168.0.3      12:45    0.00s  0.49s  0.01s w
piccardi pts/1    :0.0             Sun12    3days  1:00   1:00  top
```

Infine oltre alle informazioni riguardo gli utenti e le attività di sistema correnti, si possono consultare quelle archiviate riguardo gli accessi passati con il comando `last`. In genere il periodo coperto è quello dell'ultimo mese, in quanto, come vedremo in sez. 3.2.4, il file `/var/log/wtmp` da cui vengono ricavate le informazioni viene *ruotato* mensilmente.

Il comando stampa una lista degli ultimi accessi avvenuti in forma analoga a `who`, ma riporta oltre alla data di ingresso anche quella di uscita e la durata di ciascuna sessione, vengono stampate inoltre informazioni generali, come le date del riavvio del sistema; si avrà cioè qualcosa del tipo:

```
piccardi@anarres:~$ last
piccardi pts/0       192.168.0.3      Thu Mar 27 12:45   still logged in
piccardi pts/0       192.168.0.3      Wed Mar 26 22:22 - 01:29  (03:07)
piccardi pts/0       192.168.0.3      Wed Mar 26 12:11 - 16:20  (04:09)
piccardi tty1                         Wed Mar 26 00:02   still logged in
...
reboot   system boot  2.6.17-12-powerp Sun Mar 23 12:05 - 15:03 (4+02:57)
...
wtmp begins Fri Mar  2 17:17:47 2007
```

Il comando prevede varie opzioni, come `-n` seguita da un numero, che consente di indicare quante righe stampare o `-t` seguita da una data, che stampa lo stato a quella data; si può così vedere chi era collegato osservando le righe con `still logged in`. Al solito per una descrizione completa si faccia riferimento alla pagina di manuale.

Infine oltre `last` altri comandi che vengono in genere utilizzati per ottenere informazioni sugli accessi sono `lastb`, che stampa l'elenco dei tentativi falliti, e `uptime`, che è quello che produce la riga iniziale con i dati relativi a utenti collegati e tempo dall'ultimo avvio che viene usata come prima riga da `top` e `w`. Per tutti i dettagli come sempre si consultino le pagine di manuale.

Un ultimo comando che permette di ottenere informazioni sul sistema è `uname`, che stampa una serie di informazioni generiche. Se invocato senza argomenti il comando si limita a stampare il nome del proprio kernel (vale a dire `Linux`), che non è una informazione particolarmente significativa. Più interessanti sono le informazioni relative all'architettura hardware o al tipo di processore stampabili rispettivamente con le opzioni `-m` e `-p`, mentre con `-n` si può stampare il nome assegnato alla macchina stessa (vedi anche sez. 7.4.3) e con `-a` tutte le informazioni disponibili, ottenendo qualcosa del tipo:

```
piccardi@anarres:~$ uname -a
Linux anarres 2.6.32-5-686 #1 SMP Mon Jan 16 16:04:25 UTC 2012 i686 GNU/Linux
```

L'uso più frequente del comando però è quello in cui esso viene invocato con l'opzione `-r` per ottenere soltanto la versione del kernel corrente (nel caso `2.6.32-5-686`), informazione che viene usata spesso nella gestione dei moduli dello stesso (torneremo su questo argomento in sez. 5.2). Al solito l'elenco completo delle opzioni e tutti i dettagli del comando si possono ottenere consultando la relativa pagina di manuale.

2.4.5 I comandi per le informazioni diagnostiche

Tratteremo in quest'ultima sezione una serie di comandi che consentono di ottenere informazioni dettagliate di varia natura relativamente ai processi e alle caratteristiche del sistema, riuniti

per la loro caratteristica comune di essere utilizzati principalmente a scopi diagnostici. L'uso di questi comandi è in genere molto specialistico e la loro comprensione completa richiede una conoscenza approfondita del sistema, in particolare i concetti relativi alle reti fanno riferimento ad argomenti che saranno trattati solo a partire dal cap. 7. Per questo ne daremo solo una descrizione superficiale, da tenere come riferimento per una lettura più approfondita delle rispettive pagine di manuale, che possa però risultare utile per fornire una panoramica su come utilizzarli per effettuare la diagnosi di problemi e malfunzionamenti del sistema.

Il primo comando che prendiamo in considerazione è `dmesg`, che invocato semplicemente consente di stampare a video il contenuto del *buffer circolare*[41] dei messaggi di notifica del kernel. L'uso del comando consente di visualizzare il contenuto del buffer, che riporta tutto quanto viene scritto dal kernel, e può essere molto utile per diagnosticare le cause di alcuni problemi (ad esempio il kernel riporta eventuali errori dovuti a problemi hardware).

In generale queste stesse informazioni vengono inviate al sistema del *syslog* (sulla *facility* `kern`, vedi sez. 3.2.3) per una registrazione permanente, ma `dmesg` consente una visualizzazione immediata del buffer dei messaggi anche all'utente ordinario, anche se copre soltanto un periodo limitato, avendo il buffer circolare una dimensione prefissata. Con tutti i kernel attuali (è stata introdotta nel 2.4.23) la dimensione ha un default di 16kb, che può essere cambiato con una opzione di compilazione. Inoltre il comando consente di ottenere le informazioni anche quando il sistema del *syslog* non è attivo, come può accadere se il sistema ha problemi in fase di avvio e non lancia il relativo servizio.

Il comando prevede tre opzioni, utilizzabili solo dall'amministratore; con `-c` si cancella il contenuto del buffer dopo averlo stampato, con `-s` seguito da un valore numerico se ne reimposta la dimensione di lettura (per leggere interamente un buffer più lungo degli ordinari 16kb) ed infine con `-n` si specifica la priorità a partire dalla quale vengono mostrati i messaggi sulla console. Per ulteriori dettagli si consulti al solito la pagina di manuale.

Un altro comando molto utile è `lsof`, che permette di ottenere la lista di tutti file descriptor aperti all'interno di tutti i processi attivi nel sistema. Si tenga presente che trattandosi della lista dei file descriptor, saranno stampati non soltanto quelli associati a degli ordinari file di dati, ma anche quelli relativi agli altri tipi di file (quelli di tab. 1.1) come fifo, socket (compresi quelli associati alla connessioni di rete) ed anche alle directory, compresi quelli associati alla directory radice ed alla directory di lavoro di ciascun processo.

Il comando può essere usato anche da un utente normale, ma per poter avere tutte le informazioni occorre utilizzarlo con privilegi amministrativi. Se invocato senza nessuna opzione o argomento il comando stampa la lista di tutti i file descriptor aperti, riportando in colonna per ciascuno di essi una serie di informazioni, illustrate nell'elenco di tab. 2.41.

Il comando supporta una enorme quantità di opzioni che consentono di controllare e selezionare in maniera dettagliata sia le informazioni da stampare che quali file descriptor analizzare, per le quali si rimanda alla lettura della voluminosa pagina di manuale. Vale comunque la pena citare `-u` che consente effettuare una selezione per utente, da specificare come parametro dell'opzione, e `-p` che consente una selezione per PID, di nuovo da specificare come parametro, così da ottenere la lista dei file aperti da un processo. La selezione poi può in entrambi i casi essere invertita apponendo un carattere "`^`" al parametro.

[41] si chiama *buffer circolare* una zona di memoria di lunghezza fissa in cui salvare un ammontare limitato dei dati, una volta esaurito il buffer i nuovi dati vengono riscritti a partire dalla cima del buffer sovrascrivendo quelli precedenti, da cui deriva appunto il nome di "*circolare*".

Valore	Significato
`COMMAND`	Il nome del comando (i primi 9 caratteri) associato al processo.
`PID`	Il PID del processo.
`USER`	Il nome dell'utente (o il suo *user ID*) a cui appartiene il processo.
`FD`	per i file aperti direttamente all'interno del processo il numero associato al file descriptor seguito da una delle lettere `r` (lettura), `w` (scrittura) e `u` (lettura/scrittura) per indicare la modalità in cui è aperto; qualora si faccia riferimento ad un file descriptor non aperto esplicitamente dal processo uno dei valori di tab. 2.42.
`TYPE`	il tipo di file (quello di tab. 1.1) identificato con le sigle `REG` per i file regolari, `DIR` per le directory, `LINK` per i link simbolici, `CHR` per i dispositivi a caratteri, `BLK` per i dispositivi a blocchi, `FIFO` per le fifo e le pipe; i socket sono invece riportati per tipo (`unix` per quelli locali, `Ix4` per quelli di rete); esistono poi valori speciali come `DEL` per un file che è stato cancellato o `NOFD` per un file che non si è potuto aprire.
`DEVICE`	identificativo del dispositivo su cui si trova il file, con i valori di *major number* e *minor number*.
`SIZE`	dimensione del file, se esistente, altrimenti vuoto.
`NODE`	numero di *inode* per i file e gli altri oggetti del filesystem, nome del protocollo per i socket di rete.
`NAME`	nome del file, se questo è presente sul filesystem, le parole chiave `pipe` e `socket` per indicare file descriptor associati a *pipe* o *socket* anonimi, e gli estremi della connessione per i socket di rete.

Tabella 2.41: Informazioni di default riportate dall'output di `lsof`.

Infine se si passano uno o più argomenti al comando questi vengono normalmente interpretati come pathname di file, e verranno riportati solo i risultati dei file descriptor ad essi corrispondenti. Fa eccezione a questa regola il caso in cui il pathname specifica il *mount point* di un filesystem (vedi sez. 5.1.3) nel qual caso verranno mostrati tutti i file descriptor relativi a tale filesystem.

Valore	Significato
`cwd`	directory di lavoro del processo.
`rtd`	directory radice del processo.
`txt`	segmento di testo (codice o libreria) del processo.
`mem`	file mappato in memoria.

Tabella 2.42: Principali valori del campo `FD` dell'output di `lsof`.

Uno degli usi più comuni di `lsof` in ambito diagnostico è quello che consente di determinare quale processo stia mantenendo occupato un filesystem che si deve smontare (vedi sez. 5.1.3), sia perché c'è un file aperto su di esso, sia perché un processo ha la directory di lavoro corrente all'interno dello stesso. Un esempio di questo utilizzo è il seguente:

```
root@monk:~# lsof /mnt/
COMMAND  PID     USER   FD   TYPE DEVICE SIZE/OFF  NODE NAME
bash    9040 piccardi  cwd    DIR   8,65     4096 20288 /mnt
```

Come complemento e alternativa a `lsof` si può usare il comando `fuser`, che mostra quali processi stanno usando un certo file, socket o filesystem. Nell'uso più comune il comando richiede che si indichino file e directory di cui si vogliono conoscere eventuali utilizzatori come argomenti,

stampando per ciascuno l'elenco dei PID dei processi che li stanno utilizzando seguito da una o più lettere che ne indicano il tipo di utilizzo, secondo quanto illustrato in tab. 2.43.

Valore	Significato
c	directory di lavoro del processo.
e	eseguibile in esecuzione.
f	file aperto (non mostrato di default).
F	file aperto in scrittura (non mostrato di default).
m	file mappato in memoria.
r	directory radice del processo.

Tabella 2.43: Lettere usate da `fuser` per indicare il tipo di accesso ad un file.

Dato il risultato ordinario è poco leggibile è normale usare l'opzione `-v` per ottenere una stampa più estesa, in cui viene usata una riga per ogni processo rilevato, che riporta nell'ordine: l'utente per conto del quale viene eseguito il comando, il PID del processo, la modalità di accesso (con i valori sopra menzionati) ed il comando con cui il processo è stato eseguito. Si può inoltre usare l'opzione `-k` per inviare un segnale ai suddetti processi, il default è `SIGKILL`, ma se può specificare un altro indicandolo una ulteriore `-`*`SIGNAL`*.

Si può inoltre ottenere l'elenco di tutti i processi che usano un filesystem usando l'opzione `-m`, usando come argomento un qualunque file che stia su quel filesystem (anche se è più comune usarne la directory *mount point*). Infine il comando consente, con l'opzione `-n`, di ottenere informazione anche sui socket di rete, in tal caso occorre passare all'opzione l'ulteriore argomento `tcp` o `udp` per indicare il tipo di protocollo usato dal socket ed usare come argomento un numero di porta, seguito eventualmente, separati da virgole, dall'indirizzo locale e remoto del socket stesso (per i concetti relativi alle reti si rimanda al cap. 7).

Un altro comando diagnostico estremamente potente è `strace`, che stampa sullo *standard error* tutte le *system call* eseguite da un programma durante la sua esecuzione, consentendo così, come suggerisce il nome, di tracciare le operazioni effettuate dal programma. Ad esempio se usiamo `strace` con il comando `cat`, si avrà qualcosa del tipo:[42]

```
piccardi@anarres:~/truedoc/corso$ strace cat corso.tex > /dev/null
execve("/bin/cat", ["cat", "corso.tex"], [/* 33 vars */]) = 0
...
open("corso.tex", O_RDONLY|O_LARGEFILE) = 3
fstat64(3, st_mode=S_IFREG|0644, st_size=2315, ...) = 0
read(3, "%% corso.tex\n%%\n%% Copyright (C) "..., 4096) = 2315
write(1, "%% corso.tex\n%%\n%% Copyright (C) "..., 2315) = 2315
read(3, ""..., 4096)                    = 0
close(3)                                = 0
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
```

Una spiegazione completa di questo estratto, da cui per semplificare la lettura si sono eliminate tutte le operazioni di inizializzazione successive la chiamata iniziale a `execve` che è quella che mette in esecuzione `cat`, eccede ovviamente quanto sia possibile affrontare in questa sede,

[42] si noti la redirezione dello *standard output*, che è stata eseguita per evitare di mescolare il risultato del comando con quello di `strace`.

dato che necessiterebbe di una spiegazione dettagliata del significato delle varie *system call*, argomento che attiene alla programmazione, e non alla amministrazione di sistema.[43] È comunque possibile fare una lettura superficiale che ci mostra come prima venga aperto il file `corso.tex` sul file descriptor 3 (il risultato della `open`), e poi venga letto da questo il contenuto (con la `read`), che viene immediatamente dopo scritto (con la `write`) sul file descriptor 1, che abbiamo già detto essere appunto quello associato con lo *standard output*.

Anche solo questo esempio ci consente di vedere come `strace` consenta di analizzare le richieste fatte da un programma, ed in particolare è molto utile vedere quali file il programma cerca di aprire (con la funzione `open`). In questo modo diventa possibile capire eventuali cause di malfunzionamenti dovuti alla ricerca di file assenti o diversi da quelli che si supponevano essere quelli effettivamente utilizzati, o vedere in quale operazione può risultare bloccato un certo programma.

In genere, come nell'esempio precedente, si usa `strace` passandogli come argomento il comando che si vuole analizzare (e le eventuali opzioni ed argomenti dello stesso), ma è possibile anche tracciare un programma già in esecuzione usando l'opzione `-p`, che prende come parametro il PID del processo che si vuole analizzare. Inoltre con `-o` si può far scrivere i risultati su un file qualunque (passato come parametro) invece che sullo *standard error*, e se il nome del file inizia con una "`|`" i risultati saranno inviati sullo *standard input* del comando scritto di seguito.

Si possono infine selezionare le *system call* che si vogliono tracciare con l'opzione `-e`, cui far seguire un parametro in cui le si elencano nella forma `trace=`*`sysc1`*`,`*`sysc2`*`,...`, e si può fare una selezione negativa apponendo alla lista il carattere "`!`". Come `lsof` anche `strace` fornisce numerose opzioni che permettono di controllare e selezionare le informazioni da stampare, per le quali si rimanda alla lettura della pagina di manuale.

Un altro comando utile a scopi diagnostici è `strings` che consente di effettuare una ricerca sulle stringhe di testo presenti all'interno di un file binario e stamparle sullo *standard output*. Il comando è particolarmente utile quando si vogliono cercare le stringhe presenti all'interno dei comandi ed in particolare può aiutare ad identificare, dai relativi pathname, i file di configurazione che questo usa di default, con qualcosa del tipo:

```
strings /usr/bin/crontab | grep /etc
```

Il comando al solito prende varie opzioni che consentono di controllarne il comportamento, come `-e` che consente di specificare una codifica, o `-n` che consente di indicare la dimensione minima della stringa identificata da una sequenza di caratteri stampabili di dimensione superiore al valore specificato da tale opzione (il default è 4). Al solito per i dettagli si faccia riferimento alla pagina di manuale.

2.4.6 I comandi per la verifica di integrità

Si sono raccolti in questa sezione i programmi che vengono utilizzati per verificare l'integrità e l'autenticità di un file. Il compito come vedremo in realtà è tutt'altro che banale, in particolare per quanto riguarda la definizione di ciò che si intende per "*autentico*".

Una prima serie di questi comandi serve a stabilire se un file non presenta modifiche rispetto ad un altro, considerato come originale, verifica che in genere si effettua dopo averlo scaricato.

[43] gli interessati a questo argomento possono al solito fare riferimento a [GaPiL], il cui scopo è proprio quello di documentare adeguatamente le interfacce di sistema di Linux.

La verifica viene fatta attraverso una cosiddetta "*somma di controllo*" o *checksum* associata al file originale, che consente di controllare se la copia ottenuta è identica.

Una *checksum* non è altro che una opportuna funzione matematica che ha la caratteristica di dare risultati molto diversi anche per piccole differenze nell'input, pertanto si diventa in grado di rilevare subito eventuali modifiche. Il concetto di base è che la *checksum* viene calcolata con il file originale, che si considera come copia di riferimento, e distribuita con esso. Scaricando (o ottenendo altrimenti una copia) si potrà verificare, ricalcolando su di essa la somma, che questo porti allo stesso risultato, concludendo che il file è identico.

Queste funzioni vengono anche chiamate funzioni di *hash* in quanto una richiesta comune è che possano dare risultati diversi in presenza di dati in ingresso diversi di dimensioni qualunque, producendo comunque un valore numerico di dimensione fissa, detto appunto *hash*. Questo poi deve avvenire con probabilità trascurabile di "*collisioni*", cioè del caso in cui due input diversi generano lo stesso *hash*. Si parla poi di *hash crittografico* quando noto un *hash* è sostanzialmente impossibile ricostruire il dato che l'ha generato o trovare un dato alternativo che generi lo stesso valore (per una trattazione dell'argomento si consulti sez. 1.2.4 di [SGL]).

Il programma più semplice che si può usare a questo scopo è `cksum`, che per il calcolo della somma usa un algoritmo chiamato CRC, molto semplice e veloce, ma che non ha nessuna delle caratteristiche di un *hash crittografico*, in quanto è relativamente semplice generare file che portino allo stesso risultato data una *checksum* qualunque e le collisioni sono abbastanza frequenti. Il comando serve solo per una verifica minimale, e si usa invocandolo su uno o più file, nel qual caso per ciascuno di essi stamperà a video una riga con il valore del CRC, seguito dalla dimensione e dal nome del file. Se non si specifica nessun file il programma viene eseguito sullo *standard input*.

Se però si vuole avere la sicurezza che nessuno abbia modificato un file di cui si possiede una *checksum* occorrerà usare un programma diverso, dato che `cksum` non fornisce le garanzie di un *hash crittografico*, in quanto è relativamente semplice modificare un file e mantenere identico il valore del suo CRC. Questo rende inadeguato il programma ad una verifica di una eventuale compromissione dei propri dati o programmi da parte di un attaccante malevolo.

Per poter garantire l'impossibilità di avere una modifica che sfugga ad un controllo occorre allora che l'algoritmo utilizzato garantisca le caratteristiche di un *hash crittografico*, e cioè sostanziale impossibilità di generare lo stesso *hash* dell'originale in un file modificato. A lungo il programma usato per fare questo è stato `md5sum`, che usa per il calcolo un algoritmo chiamato MD5, considerato in grado di rispondere al requisito.

Anche in questo caso il programma richiede che si specifichi come argomento uno o più file da controllare, stampando per ciascuno di essi una riga con il valore dell'*hash* MD5 (un numero a 128 bit in forma esadecimale) seguito dal nome del file. Di nuovo se non si specifica nessun file il programma opera sullo *standard input*. Un esempio di uso del programma è il seguente:

```
piccardi@monk:~/truedoc/sicurezza$ md5sum *.tex
17d80d7547d6a7bbc2d84bfaec9ee5f2  cryptoapp.tex
9f3dfec4f383eee3e75a374c5f07a056  firewall.tex
4b10bef0b1ada92990ed190a693b3bb7  gpg.tex
be501bad4f38ce267c227f10788f16db  ids.tex
3a3c65cfe2b1c794efbc6e8b70969ccb  introsec.tex
ac29b2b82fc88ffaa12f7aa4920f412a  sicurezza.tex
```

Si può inoltre utilizzare l'opzione `-c`, che consente di indicare come unico argomento un file che contenga una lista di risultati precedenti. Il comando verrà applicato a ciascuno dei file elencati, segnalando eventuali differenze, diventa così possibile effettuare velocemente un controllo di integrità se ci si è salvati da qualche parte l'uscita di una invocazione precedente.

Da qualche anno però sono stati scoperti seri problemi nell'algoritmo MD5, benché resti ancora difficile ottenere uno stesso *hash* con modifiche arbitrarie, è diventato possibile generare delle collisioni e per questo oggi l'algoritmo non viene più considerato sicuro anche se la complessità di un attacco mirato a mascherare una modifica ad un file resta abbastanza alta. Al suo posto comunque viene consigliato un algoritmo alternativo, lo SHA.

In realtà in questo caso si tratta di una famiglia di algoritmi, in quanto la dimensione dell'*hash* è variabile, ed esistono varie versioni dell'algoritmo a partire dai 160 bit della prima versione, denominata SHA1. Con la versione successiva, chiamata genericamente SHA2, sono inoltre possibili varianti con dimensioni diverse dell'*hash* che vengono identificate apponendo al nome SHA il numero di bit usati per l'*hash* (224, 265, 384, 512). Benché siano state trovate delle debolezze nell'algoritmo SHA1, queste al momento prospettano solo una riduzione della complessità nella generazione delle collisioni.

Le stesse debolezze sono al momento solo teorizzate per SHA2 per la similarità dell'algoritmo, ma senza nessun risultato concreto. Trattandosi comunque di debolezze che riducono la complessità di calcolo nella ricerca di una possibile collisione, esiste comunque la possibilità di riportarla ad un valore che renda impraticabile qualunque attacco semplicemente usando un *hash* di dimensioni superiori. Per questo motivo l'algoritmo ad oggi viene comunque considerato sicuro, e nella versione a 512 bit viene dichiarato adeguato anche per l'impiego in condizioni di massima sicurezza.

Il comando che si può utilizzare per creare e verificare delle *checksum* SHA è `shasum`, che di default usa l'algoritmo SHA1. Con l'opzione `-a` si può però richiedere l'uso di una qualunque delle versioni alternative, indicando il relativo numero di bit.[44] Il programma usa gli argomenti allo stesso modo di `md5sum`, e produce un risultato analogo, si avrà così:

```
piccardi@monk:~/truedoc/sicurezza$ shasum *.tex
5f4c0cc0be9b6ae0331b45c7feef861a3bcf1caf  cryptoapp.tex
938541121a926948ff1f2125d36837177882b6b6  firewall.tex
a4f0609fc2309dea6e75bad0a959f34262e9fd64  gpg.tex
9437d7bc0e73e695de7a567eef527af4f5c14e4b  ids.tex
20f70ca35a77d10c3c9d769b2381b7c783a1961b  introsec.tex
3548297780fa69bdf0a8f91b184993d835465e52  sicurezza.tex
```

Il comando riconosce anche l'opzione `-c` con lo stesso significato di `md5sum`, se allora si salva il precedente risultato con un comando come `shasum *.tex > lista.sha`, si potrà verificare che nessun file è stato modificato con:

```
piccardi@monk:~/truedoc/sicurezza$ shasum -c lista.sha
cryptoapp.tex: OK
firewall.tex: OK
gpg.tex: OK
ids.tex: OK
introsec.tex: OK
sicurezza.tex: OK
```

[44] in realtà il comando `shasum` è un *wrapper* che invoca una serie di altri comandi (`sha1sum`, `sha224sum`, `sha256sum`, ecc.) a seconda della versione scelta.

Se non si vuole che vengano ristampate le verifiche ma che il comando riporti nello stato di uscita la corrispondenza o meno della verifica, si può usare l'opzione `-s`, per tutti dettagli e l'elenco completo delle opzioni si consulti la pagina di manuale.

Benché i comandi precedenti consentano di verificare l'integrità di un file una volta che si disponga della relativa *checksum*, questo non ci aiuta se non siamo in grado di stabilire se anche quest'ultima è "*autentica*". Il punto cruciale è che se un attaccante può sostituire un file, ad esempio quello di un pacchetto che si vuole installare, con una sua copia opportunamente "*modificata*", potrà anche generare la relativa *checksum*, e la verifica della stessa diventa inutile.

I comandi illustrati possono servire allora a rilevare eventuali modifiche rispetto a file dei quali si è ottenuta, in maniera indipendente e verificata, una *checksum* che si sa essere quella giusta. L'unico modo per assicurarsi, quando si scarica un file dalla rete, che la relativa *checksum* non sia stata creata ad arte, è quello di ricorrere alla cosiddetta *firma digitale* ed alla crittografia a chiave asimmetrica. L'argomento va al di là dello scopo di questo testo (ma è trattato in dettaglio nella sez. 1.2 e nel capitolo 2 di [SGL]), ma il concetto di base è che è possibile apporre firme digitali su un file, che non solo attestano che questo non è stato modificato, ma anche che il file stesso, o la *checksum* che vi si è allegata, è proprio quella generata dall'autore della firma digitale.

Benché esistano diverse possibilità, il modello utilizzato nel mondo GNU/Linux per ottenere questo tipo di garanzia, ed utilizzato da tutte le distribuzioni per garantire ad esempio l'autenticità dei propri pacchetti software, è quello di firmare i singoli pacchetti, gli hash crittografici, e nel caso di Debian pure il contenuto generale dell'archivio, con il programma GPG. Non tratteremo in questa sede i dettagli del modello del *Web of trust* e le caratteristiche generali di GPG, (che sono affrontati in maniera completa in sez. 2.2 di [SGL]), limitandoci ad esporre alcuni concetti in maniera molto generica ed a riprendere da quella sezione solo la parte relativa alla sintassi del comando.

Il programma, il cui nome è l'acronimo di *GNU Privacy Guard* è una reimplementazione di un programma, PGP (acronimo di *Pretty Good Privacy*), nato per cifrare e firmare i messaggi di posta elettronica. Il programma prevede che un utente venga identificato da una coppia di chiavi, cui vengono associati uno o più identificativi, in genere espressi nella forma di un indirizzo di posta elettronica. La chiave pubblica viene distribuita e può essere usata da chiunque per creare messaggi cifrati che possono essere solo decifrati dalla chiave privata, che invece deve essere mantenuta e protetta con la massima cura, in quanto oltre a decifrare può anche essere utilizzata per realizzare una firma digitale.

In genere il programma viene usato per corrispondere, nel qual caso ci si procura la chiave pubblica del destinatario (esistono anche dei *keyserver* che raccolgono le chiavi pubbliche di coloro che le vogliono pubblicare) e si cifra il messaggio con quella in modo che solo lui possa decifrarlo. Ma si può usare il programma anche per cifrare i propri file, usando la propria chiave privata, in modo da poter essere i soli a decifrarli.

Un impiego di questo tipo potrebbe essere quello in cui si cifrano dei file (ad esempio un backup) prima di inviarli su un server pubblico, il vantaggio in questo caso è che, non essendo la propria chiave privata necessaria per cifrare, questa può essere mantenuta in sicurezza fuori dal computer su cui si cifra il file, ed una compromissione dello stesso non comporta la possibilità di decifrare quanto è stato cifrato in precedenza (cosa che per un backup però avrebbe poco senso, essendo in tal caso disponibili i file originali).

Oltre che cifrare i file GPG consente anche di firmarli, se infatti si genera un *hash crittografico* di un file e lo si cifra con la propria chiave privata si ottiene appunto una firma digitale. In tal caso infatti una persona in possesso della nostra chiave pubblica (potenzialmente chiunque) potrà decifrarlo. Ma dato che solo con la chiave privata si può generare un messaggio decifrabile con la chiave pubblica, si dimostra che chi ha creato quella "firma" possiede la chiave privata (e se ne certifica l'identità), inoltre con l'*hash crittografico* della firma si potrà verificare anche che il file è integro.

In questo caso, anche se un attaccante potesse sostituire un eventuale pacchetto software, la sua *checksum*, e la firma digitale non riuscirebbe comunque a passare inosservato. Infatti chiunque abbia avuto cura di procurarsi in anticipo la chiave pubblica con la quale lo sviluppatore firma il pacchetto garantendone autenticità ed integrità, rileverebbe immediatamente che la firma non corrisponde a quella aspettata, dato che solo la chiave privata dello sviluppatore può generare una firma digitale "*corretta*".

Si noti comunque che anche in questo caso la sicurezza non può essere assoluta, la chiave privata dello sviluppatore potrebbe essere stata compromessa, ottenendo la possibilità di generare una firma corretta, oppure qualcuno potrebbe averci ingannati riuscendo a spacciare la sua chiave pubblica per quella dello sviluppatore, e con questo la possibilità di farci credere che la firma sia corretta. Ma la situazione resta comunque nettamente migliore rispetto al precedente caso di controllo basato sulla sola verifica di un *hash crittografico*.

Si noti come in questo caso sia critico, come lo sarà per le chiavi pubbliche di SSH (vedi sez. 8.3) accertarsi che le chiavi pubbliche che si utilizzano siano effettivamente quelle "*giuste*". In questo viene in aiuto il modello del *web of trust* che GPG utilizza, per il quale di nuovo si rimanda alla lettura della sez. 2.2 di [SGL].

Benché esistano diversi programmi di gestione dotati di interfaccia grafica, noi tratteremo soltanto la versione originale di GPG a riga di comando, `gpg`. Il programma è estremamente vasto e complesso, le varie operazioni che può eseguire vengono attivate tramite un lunghissimo elenco di opzioni la gran parte delle quali deve essere specificata in formato esteso.

Alcune di queste costituiscono dei veri e propri sottoprogrammi e devono essere usate singolarmente, attivando speciali operazioni interattive ed anche una riga di comando interno. Altre controllano le operazioni più ordinarie e possono richiedere o meno l'indicazione di un argomento al programma, il cui significato varia a seconda dell'opzione stessa. Infine alcune possono essere usate in combinazione con le altre per controllare le modalità di funzionamento del programma.

Si sono riportate le opzioni più comuni nell'uso ordinario del programma, che in genere sono quelle per cui è presente anche una versione in forma non estesa, in tab. 2.44. Di tutte le altre tratteremo qui soltanto le più rilevanti, mentre per gli approfondimenti e tutti i dettagli si rimanda alla abbondante documentazione disponibile a partire dalla pagina di manuale.

Il primo passo per l'uso di GPG è quello di generare una propria coppia di chiavi. Questa è in genere una operazione da eseguire una volta per tutte, e non rientra nell'uso ordinario del programma. Per farlo occorre invocare `gpg` con l'opzione `--gen-key` che fa partire la procedura interattiva in cui vengono richieste tutte le informazioni necessarie. Dato che il programma è di uso personale, questo deve essere usato come utente normale, e tutti i suoi dati verranno mantenuti nella directory `.gnupg` della propria home directory.

Questo significa anche che non è il caso di usarlo su macchine di cui non si è il solo amministratore, perché per quanto le chiavi private siano protette da una *passphrase* esistono

molti modi (ad esempio l'installazione di una versione "*taroccata*" del programma) con cui un amministratore potrebbe impossessarsi della vostra chiave privata.

Un esempio di creazione delle proprie chiavi, fatto passo per passo, è quello che riportiamo di seguito. Anzitutto occorre scegliere algoritmi di crittografia, dimensione delle chiavi e scadenza:

```
$ gpg --gen-key
gpg (GnuPG) 1.4.11; Copyright (C) 2010 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y
```

ed in questo caso si sono accettati i default per le impostazioni e si è risposto di sì alla domanda finale (in caso di errore in una delle scelte si può ripetere la procedura rispondendo no), una volta fatto questo bisogna scegliere un opportuno *User ID*:

```
You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Truelite Srl
Email address: info@trl.tl
Comment: Truelite
You selected this USER-ID:
    "Truelite Srl (Truelite) <info@trl.tl>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
```

dove vengono richieste le varie informazioni che ne fanno parte. Una volta completata la fornitura delle informazioni di identificazione verrà chiesta una *passphrase* per proteggere la propria chiave privata. Dato che la *passphrase* costituisce l'unica difesa contro un uso non autorizzato della chiave (in caso di compromissione o perdita della macchina su cui si lavora) è opportuno che sia scelta con molta cura e sia sufficientemente lunga e complessa; non venendo visualizzata la si dovrà immettere due volte per conferma:

```
You need a Passphrase to protect your secret key.

Enter passphrase:
Repeat passphrase:
```

una volta fatto questo il programma provvederà alla generazione della chiave privata, per farlo è necessario avere una quantità sufficiente di numeri casuali, cosa che in genere causa la richiesta di generare sufficiente entropia nel sistema con varie azioni,[45] una volta ottenuta la quale verranno creati i dati necessari e stampato un riassunto conclusivo:

```
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

Not enough random bytes available.  Please do some other work to give
the OS a chance to collect more entropy! (Need 144 more bytes)
....+++++
..........+++++
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

Not enough random bytes available.  Please do some other work to give
the OS a chance to collect more entropy! (Need 24 more bytes)
..+++++
..+++++
gpg: /home/piccardi/.gnupg/trustdb.gpg: trustdb created
gpg: key 58AF19F7 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
pub   2048R/58AF19F7 2011-07-07
      Key fingerprint = 3A62 34E7 E2EB E1DA 3EF5  9D60 66F3 F29A 58AF 19F7
uid                  Truelite Srl (Truelite) <info@trl.tl>
sub   2048R/F1493ABF 2011-07-07
```

Si tenga conto che `gpg` consente di gestire anche più coppie di chiavi; se pertanto si vogliono usare due chiavi diverse per due identità diverse si potrà ripetere l'operazione di generazione delle chiavi una seconda volta, e la nuova chiave verrà aggiunta nel proprio *keyring* (il "*portachiavi*") all'interno di `.gnupg`. È sempre possibile avere due chiavi con riferimento ad un *User ID* identico, in caso di ambiguità infatti si potrà fare comunque riferimento al *Key-ID* per indicare quale delle chiavi si vuole usare.

Un secondo passo iniziale, che si consiglia di mettere in pratica subito dopo la creazione di una coppia di chiavi, è quello di generare un certificato di revoca. Questa è una misura di precauzione nei confronti dell'eventualità della perdita totale della chiave privata, (come lo smarrimento o il furto dell'unico computer su cui la si teneva) o di impossibilità di accesso alla stessa, (ad esempio se non ci si ricorda più la passphrase) che permette di invalidare permanentemente la propria chiave PGP anche quando questa non è più disponibile. È ovviamente sempre possibile generare in un secondo tempo il certificato in caso di perdita o compromissione della propria chiave, ma per farlo occorre averne a disposizione almeno una copia.

[45] la pressione di tasti fa riferimento alla tastiera fisica, se si esegue il comando in remoto con una sessione `ssh` la cosa non avrà alcun effetto (la rete non è considerata una fonte affidabile di rumore).

Ovviamente occorrerà proteggere adeguatamente tale certificato di revoca, perché se questo pervenisse in mano ad un terzo ostile questi potrebbe invalidare la chiave a cui fa riferimento. La sua perdita però è meno dannosa rispetto a quella della chiave stessa, in quanto il suo scopo è solo quello della revoca, e quindi non è possibile usarlo per impersonare il titolare della chiave.

Per la generazione del certificato di revoca occorre usare l'opzione `--gen-revoke` che, come molte altre opzioni di `gpg`, richiede che si specifichi come argomento del comando la chiave a quale si fa riferimento. Questo si può fare in due modi, si può indicare il valore in esadecimale del *Key-ID* della chiave scelta, o si può indicare una stringa il cui testo verrà usato per individuare la chiave con una ricerca nel testo degli *User ID* ad esse associati; nel caso di corrispondenze multiple verrà sempre selezionata la prima, per evitare le ambiguità occorre usare il *Key-ID*.

Un esempio della generazione del certificato di revoca, in cui si è usata anche l'opzione `--output` per indicare un file di uscita (altrimenti il testo del certificato sarebbe stato stampato sullo *standard output*), è il seguente:

```
$ gpg --output revoca.asc --gen-revoke Truelite
sec  2048R/58AF19F7 2011-07-07 Truelite Srl (Truelite) <info@trl.tl>

Create a revocation certificate for this key? (y/N) y
Please select the reason for the revocation:
  0 = No reason specified
  1 = Key has been compromised
  2 = Key is superseded
  3 = Key is no longer used
  Q = Cancel
(Probably you want to select 1 here)
Your decision? 0
Enter an optional description; end it with an empty line:
> Revoca precauzionale
>
Reason for revocation: No reason specified
Revoca precauzionale
Is this okay? (y/N) y

You need a passphrase to unlock the secret key for
user: "Truelite Srl (Truelite) <info@trl.tl>"
2048-bit RSA key, ID 58AF19F7, created 2011-07-07

ASCII armored output forced.
Revocation certificate created.

Please move it to a medium which you can hide away; if Mallory gets
access to this certificate he can use it to make your key unusable.
It is smart to print this certificate and store it away, just in case
your media become unreadable.  But have some caution:  The print system of
your machine might store the data and make it available to others!
```

Una volta fatto questo si otterrà all'interno del file `revoca.asc` (la convenzione è quella di usare l'estensione `.asc` per i file che contengono rappresentazioni testuali delle chiavi PGP) si otterrà un contenuto del tipo:

```
revoca.asc
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.11 (GNU/Linux)
```

```
Comment: A revocation certificate should follow

iQEzBCABAgAdBQJOAKulFh0AUmV2b2NhIHByZWNhdXppb25hbGUACgkQm3VwX32z
UE0hNwgA2/+LV+6QGlVrDBRgFwoEoKgq378s9S6Jy0hC/Yru52DnAVgV/SoUvTj3
...
jIG1qsLdzLJCNzNf+Rkyh6ywGsOjWw==
=g2AK
-----END PGP PUBLIC KEY BLOCK-----
```


e si noti come in precedenza il comando ci abbia notificato che è stata forzata una uscita in formato testuale (il cosiddetto *ASCII armored*); questo rende possibile anche una semplice stampa del certificato.

Una volta che si disponga di una propria coppia di chiavi si dovrà iniziare ad utilizzarla. Perché altri ci possano inviare messaggi cifrati dovranno però disporre della nostra chiave pubblica, lo stesso vale qualora volessero firmarla dopo aver verificato la nostra identità, occorre pertanto avere un meccanismo di distribuzione.

Il modo più semplice per distribuire la propria chiave pubblica è quello di pubblicarla su un file, per questo esiste l'opzione `--export` che consente di salvare un chiave pubblica in un formato standard che la renda utilizzabile da terzi. Se usata da sola i dati verranno stampati a video in formato binario, pertanto è comune usare l'opzione `--output` (abbreviabile in `-o`) per indicare un file di destinazione e l'opzione `--armor` (abbreviabile in `-a`) per richiedere la codifica *ASCII armored*. In questo modo la chiave sarà in formato testo, che può essere facilmente scambiato per email o pubblicato, e la cui copia non richiede nulla di più di un taglia ed incolla.

Dato che come accennato un *keyring* personale può contenere diverse chiavi, sia pubbliche che private, anche `--export` richiede che si indichi quella che si vuole esportare con un *User ID* o un *Key-ID*, con le stesse convenzioni riguardo la scelta *User ID* sulla base della stringa passata come argomento già illustrate per `--gen-revoke`. Un esempio di esportazione è allora il seguente:

```
$ gpg --output key.gpg --armor --export info@trl.tl
```

e si otterrà un contenuto di `key.gpg` del tipo:

key.gpg


```
 -----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.11 (GNU/Linux)

mQENBE4ApmQBCADv7AuurV1SnpOtylZazkl39QOAQdMhGyPtr8AVFJOKVd8qr5Da
...
awgseHHXiHZD2Mdc7p5SQTjqzWTW1ZcM1zItbpkt8xjcXCdCX8NhXyQT2msqKyGT
=TwxL
-----END PGP PUBLIC KEY BLOCK-----
```


Una volta ottenuto il file con la chiave, o il relativo testo, lo si potrà inviare ai propri corrispondenti, ma una delle funzionalità più interessanti di `gpg` è che il programma supporta direttamente anche l'interazione con i cosiddetti *keyserver*, una rete di server presenti su Internet e dedicati alla distribuzione delle chiavi, che possono essere interrogati direttamente tramite il comando con le due opzioni `--send-keys` e `--recv-keys`.

In particolare `--send-keys` consente di inviare ad un *keyserver* una qualunque delle chiavi pubbliche che si hanno nel portachiavi, da indicare esclusivamente per *Key-ID*. Pertanto si potrà eseguire la pubblicazione di una chiave con:

```
$ gpg --send-key 58AF19F7
```

ed il comando inserirà la chiave nel *keyserver* la prima volta, e la aggiornerà con eventuali dati (altre identità o firme ricevute) le volte successive.

Una volta che una chiave è stata pubblicata su un *keyserver* sarà possibile per chiunque scaricarla direttamente via rete, `gpg` supporta una specifica opzione, `--recv-keys`, che scarica ed importa direttamente una chiave, sempre indicata per *Key-ID*, con un comando del tipo:

```
$ gpg --recv-keys 2A972F9D
gpg: requesting key 2A972F9D from hkp server keys.gnupg.net
gpg: key 2A972F9D: public key "Simone Piccardi (Truelite SrL) <piccardi@truelite.it>" imported
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: Total number processed: 1
gpg:               imported: 1
```

Nei due esempi precedenti si è sempre usato il *keyserver* di default, `keys.gnupg.net`, ma è sempre possibile usare l'ulteriore opzione `--keyserver` per specificare un *keyserver* alternativo; l'opzione richiede l'indicazione dello stesso come parametro da specificare in forma di indirizzo IP o di nome a dominio.[46]

Se invece si dispone direttamente di un file contente una chiave la si potrà importare direttamente con l'opzione `--import`, se non si specifica altro il comando leggerà la chiave dallo standard input, altrimenti si potrà passare come argomento il file che la contiene. Pertanto se si è scaricata la chiave con cui viene firmato l'archivio di Debian Squeeze nel file `archive-key-6.0.asc`, la si potrà importare con:

```
$ gpg --import archive-key-6.0.asc
gpg: key 473041FA: public key "Debian Archive Automatic Signing Key (..." imported
gpg: Numero totale esaminato: 1
gpg:              importate: 1  (RSA: 1)
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
```

Si tenga presente che importare una chiave non significa ancora darle alcun livello di fiducia e che il fatto che sia disponibile su un *keyserver* non ha nessuna implicazione su questo piano; la chiave resterà considerata *untrusted* fintanto che non la si sia verificata esplicitamente (ed eventualmente firmata) o sia firmata da altri corrispondenti dei quali si è stabilito di avere fiducia (torneremo sulla gestione di questi aspetti più avanti).

Una volta completate le operazioni preliminari per creare la propria coppia di chiavi ed ottenere le chiavi dei corrispondenti che ci interessano, l'uso ordinario di `gpg` consiste principalmente nella ricezione e nell'invio di documenti cifrati o firmati o entrambi. Le opzioni relative a queste operazioni, insieme alle altre di più comune utilizzo, sono riassunte in tab. 2.44.

Una volta che si disponga della chiave di un corrispondente si potrà cifrare un documento per lui con l'opzione `--encrypt` (abbreviabile in `-e`); questa opzione richiede l'uso della successiva opzione `--recipient` (abbreviabile in `-r`) per indicare il destinatario che deve essere specificato con un ulteriore parametro che ne identifichi la chiave pubblica, al solito il parametro viene usato

[46] in realtà il parametro ha una forma generica più complessa: `scheme://keyserver.name.or.ip:port` dove `scheme` può assumere i valori `hkp`, `ldap` o `mailto` per l'uso di diversi schemi di comunicazione (via HTTP, LDAP o email); dato che la gran parte dei *keyserver* usa il protocollo via web, si omette `hkp://` essendo questa il default.

Opzione		Significato
`--output`	`-o`	indica di scrivere l'uscita del comando sul file passato come parametro.
`--armor`	`-a`	indica la generazione di una uscita in testo puro codificato, il cosiddetto *ASCII armored*.
`--recipient`	`-r`	indica il destinatario per cui si vuole cifrare, da indicare per *User ID* o *Key-ID*.
`--encrypt`	`-e`	esegue la cifratura del file indicato come argomento del comando (o letto dallo *standard input*).
`--decrypt`	`-d`	esegue la decifratura del file indicato come argomento del comando (o letto dallo *standard input*).
`--sign`	`-s`	crea una firma del file indicato come argomento (o letto dallo *standard input*).
`--detach-sign`	`-b`	firma il file indicato come argomento (o letto dallo *standard input*) su un file a parte.
`--local-user`	`-u`	richiede di usare la chiave privata corrispondente all'*User ID* passato come parametro.
`--verify`	–	richiede la verifica della firma.

Tabella 2.44: Principali opzioni di `gpg`.

come stringa di ricerca negli *User ID* delle chiavi pubbliche presenti nel proprio *keyring* o come indicazione del *Key-ID*.

L'opzione `-r` può essere usata più volte se i destinatari sono più di uno. Di nuovo i dati da cifrare verranno letti dallo *standard input* a meno di non indicare un file come argomento, e verranno scritti sullo *standard output* in firmato binario a meno di non usare le opzioni `-o` per indicare un file diverso e `-a` per richiedere l'*ASCII armored*.

Si tenga presente inoltre che qualora si cifri verso un corrispondente non verificato il comando stamperà le caratteristiche della chiave e chiederà se andare avanti lo stesso, fermandosi se non si da una approvazione esplicita; si otterrà pertanto un risultato del tipo:

```
$ gpg -o cifrato.gpg -a -e --recipient 26257B68 chiaro.txt
gpg: 56EC3680: There is no assurance this key belongs to the named user

pub  2048g/56EC3680 1999-09-23 Christian Surchi <christian@truelite.it>
 Primary key fingerprint: D1E2 9A9D 1712 0E94 8671  834E 0CFF 30E1 2625 7B68
      Subkey fingerprint: 4152 402F 61B0 3D3E 06A4  A587 18B4 4D99 56EC 3680

It is NOT certain that the key belongs to the person named
in the user ID.  If you *really* know what you are doing,
you may answer the next question with yes.

Use this key anyway? (y/N) y
```

e si dovrà indicare di usare la chiave esplicitamente.[47] A questo punto nel file `cifrato.gpg` si avrà il messaggio cifrato che si potrà inviare al destinatario, con un contenuto del tipo:

cifrato.gpg

```
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.4.11 (GNU/Linux)
```

[47]la cosa si può fare quando quello che interessa è la cifratura dei dati e non l'autenticità del corrispondente, per essere ragionevolmente sicuri di questa, e rimuovere l'avviso del comando, occorrerà assegnare un livello di fiducia alla chiave, cosa che vedremo più avanti.

```
hQQOA9pvZzRosObsEBAAsixRt/H5+tCoofUcpKZGCbPB/1Nfb1I9Jgp7pvHoP4xx
...
ni9WWMSr1bnA4oHEcpGNyEe4J5I8r1IF9Rm2N6Tl/J+kElg=
=4ZR7
-----END PGP MESSAGE-----
```

e soltanto il destinatario potrà decifrarlo.

Qualora invece si voglia firmare un documento, invece di cifrarlo, si hanno due possibilità: con l'opzione `--sign` (abbreviabile in `-s`) si può generare un file firmato, contenente sia il contenuto originale che la firma digitale. Spesso però è più semplice, ad esempio quando si pubblicano dei file, mettere a disposizione la firma su un file a parte con la cosiddetta *detached signature*, nel qual caso occorrerà invece usare l'opzione `--detach-sign` (abbreviabile in `-b`). Qualora si abbiano più chiavi private (relative a diverse identità) si potrà indicare quale si vuole usare con l'opzione `--local-user` (abbreviabile in `-u`). Un esempio di creazione di un file firmato è il seguente:

```
$ gpg --sign chiaro.txt
You need a passphrase to unlock the secret key for
user: "Truelite Srl (Truelite) <info@trl.tl>"
2048-bit RSA key, ID 58AF19F7, created 2011-07-07

Enter passphrase:
```

il cui risultato è creare un file binario `chiaro.txt.gpg` contenente sia il contenuto del file originario che la firma in forma di messaggio PGP. L'opzione si può specificare anche insieme a `-e` così da generare un messaggio cifrato e firmato allo stesso tempo. Se invece si vuole firmare un file e distribuire la *detached signature*, si potrà usare il comando:

```
$ gpg --armor --detach-sign chiaro.txt
You need a passphrase to unlock the secret key for
user: "Truelite Srl (Truelite) <info@trl.tl>"
2048-bit RSA key, ID 58AF19F7, created 2011-07-07

Enter passphrase:
```

e in questo caso si otterrà un file separato `chiaro.txt.asc`, in formato *ASCII armored*, contenente solo la firma nella forma:

chiaro.txt.asc

```
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.11 (GNU/Linux)

iQEcBAABAgAGBQJOFxivAAoJEGbz8ppYrxn3jOoH/i7aSgq/8PxnPIy+vjYAO+GG
...
=7ysA
-----END PGP SIGNATURE-----
```

che in questo caso dovrà essere distribuito insieme al file originario.

Per verificare una firma si deve usare invece l'opzione `--verify`, che richiede come argomento il file della firma. Se questo è un messaggio completo non occorre altro, se questo è una *detached*

signature il comando cerca il file originale sulla base del nome di quello della firma (di default viene aggiunta l'estensione `.asc` o `.sig`), togliendo l'estensione, ma si può indicare come originale un file qualsiasi specificandolo come secondo argomento. Pertanto si potrà verificare la firma del precedente file con:

```
$ gpg --verify chiaro.txt.asc
gpg: Signature made Fri Jul  8 16:48:15 2011 CEST using RSA key ID 58AF19F7
gpg: Good signature from "Truelite Srl (Truelite) <info@trl.tl>"
```

in cui il riferimento al file originale `chiaro.txt` è automatico.

Se infine si deve decifrare un messaggio che ci è stato inviato l'opzione relativa è `--decrypt` (abbreviabile in `-d`), che richiede si specifichi il file con il messaggio. Il comando decifra il messaggio e lo stampa, se questo è firmato verifica automaticamente anche la firma, posto che si abbia la chiave pubblica del mittente. Un esempio del relativo risultato è:

```
$ gpg --decrypt cifrato.gpg
You need a passphrase to unlock the secret key for
user: "Truelite Srl (Truelite) <info@trl.tl>"
2048-bit RSA key, ID F1493ABF, created 2011-07-07 (main key ID 58AF19F7)

gpg: encrypted with 2048-bit RSA key, ID F1493ABF, created 2011-07-07
      "Truelite Srl (Truelite) <info@trl.tl>"
PROVA --- PROVA --- PROVA

Prova di testo da cifrare.

PROVA --- PROVA --- PROVA
```

Come abbiamo visto in precedenza nel generare un messaggio cifrato per un corrispondente che non si conosce `gpg` richiede una conferma esplicita non potendosi considerare fidata la relativa chiave. Questa non è che la diretta conseguenza del modello del *web of trust* in cui ciascuno deve assumersi la responsabilità di decidere di chi si fida o meno.

La gestione del livello di fiducia, così come le varie operazioni di gestione delle chiavi, si effettuano con la specifica opzione `--edit-key` del comando che richiede si passi come argomento un *user ID* o un *Key-ID*. L'opzione stampa i dati riassuntivi della chiave scelta (con i vari *User ID* ad essa associati) e poi porta su una riga di comando interna da cui si possono eseguire le varie operazioni tramite altrettanti comandi. Di questi citeremo soltanto `help`, che consente di stampare una lista degli stessi, per una trattazione più dettagliata si rimanda alla parte finale della sezione 2.2.3 di [SGL].

Capitolo 3

La configurazione dei servizi

3.1 I file di configurazione

In questa sezione tratteremo la gestione generica dei file di configurazione all'interno di un sistema GNU/Linux, introducendo alcuni concetti validi in generale per qualunque file di configurazione. Descriveremo poi direttamente i file di configurazione di alcuni servizi di base, come quelli che controllano il comportamento delle librerie dinamiche, l'impostazione dei parametri del kernel e la localizzazione dei programmi.

3.1.1 Una panoramica generale

A differenza di Windows che tiene tutte le configurazioni in un unico file binario, il *registro*, le sole due caratteristiche comuni che potete trovare nei file di configurazione su un sistema GNU/Linux sono che essi sono mantenuti, come illustrato in sez. 1.2.3, nella directory `/etc/` e che sono file di testo. Questo vale in generale per tutti i file di configurazione, e non è limitato a quelli che tratteremo nel prosieguo di questa sezione.

La ragione di questa frammentazione dei file di configurazione deriva dell'architettura del sistema (illustrata in sez. 1.1), per cui tutti i servizi sono forniti da opportuni programmi, che non è affatto detto siano sempre gli stessi anche per uno stesso servizio: ad esempio esistono diversi programmi che gestiscono l'invio e la ricezione della posta elettronica, ma chiaramente se ne installerà uno solo, e si userà il file di configurazione specifico dello stesso.

Ciò comporta che i formati dei file di configurazione possano essere anche i più vari, dipendendo ciascuno dalla sintassi adottata dal relativo programma; per cui, anche se esistono delle convenzioni generali come ignorare le righe vuote o considerare il carattere "`#`" l'inizio di un commento interno al file, esiste una enorme varietà di formati, ed anche queste semplici regole non è detto che vengano sempre rispettate.

Ma anche se non esiste un formato universale, alcuni programmi han finito per l'adottare formati comuni o simili per i propri file di configurazione. Uno dei più utilizzati, che per questo vale la pena citare esplicitamente, è quello dei cosiddetti *keyfile*, una variante del formato dei file `.ini` di Windows, esteso secondo le *XDG Desktop Entry Specification* (vedi `http://standards.freedesktop.org/desktop-entry-spec/latest/`) in cui si suddivide il file di

configurazione in sezioni, introdotte ed identificate da un nome racchiuso fra parentesi quadre, in cui inserire direttive espresse in forma di assegnazione *chiave=valore*. Per i nomi delle direttive di configurazione si usano solo caratteri alfanumerici, gli spazi nelle assegnazioni vengono ignorati e normalmente viene fatta distinzione fra maiuscole e minuscole. Infine, a differenza dei file `.ini` di Windows e seguendo la tradizione Unix, viene consentito anche l'uso del carattere "`#`" oltre al "`;`" come introduzione ad una riga di commento, ignorando tutto quello che segue.

Se da una parte tutta la complessità insita in una directory contenente centinaia di file di configurazione diversi può spaventare, dall'altra ha invece il grande vantaggio che le modifiche ad un singolo file di configurazione non hanno alcun modo di influenzare quelli di altri programmi, essendo detti file tutti indipendenti. Il secondo enorme vantaggio è che i file di configurazione sono dei normali file di testo per cui è possibile effettuare ricerche ed operazioni complesse con gli ordinari comandi di shell trattati in sez. 2.2, ed eseguire le operazioni di configurazione con un editor qualunque.

Un secondo aspetto delle configurazioni di cui bisogna tenere conto è quello della multiutenza, per cui con molti programmi è possibile che ogni utente possa scegliere le impostazioni che ritiene più appropriate mettendo un ulteriore file di configurazione nella propria home directory. In genere questi file sono invisibili (iniziano cioè con un "`.`") ed hanno lo stesso nome del loro analogo di `/etc/` valido per tutto il sistema. Questa è una forma semplice e pulita di consentire a ciascun utente di personalizzare le sue scelte senza dover andare a scomodare le impostazioni generali di tutto il sistema.

È da tenere presente infine che per molti file di configurazione viene installata anche una pagina di manuale che ne spiega il formato e le direttive, accessibile usualmente con `man nomefile`, o nel caso di omonimia con un comando o una funzione di sistema, con `man 5 nomefile` (si ricordi quanto detto in sez. 2.4.2). In seguito, per i file di configurazione che prenderemo in esame, faremo una descrizione generale, trattando solo le caratteristiche principali; per questo varrà sempre la pena controllare la documentazione, che contiene tutti i dettagli.

3.1.2 La gestione delle librerie condivise

Una delle funzionalità più importanti per l'esecuzione dei programmi in un qualunque sistema operativo è quella della gestione delle librerie condivise, quelle che in Windows vengono chiamate DLL (da *Dinamically Linked Library*) e che nei sistemi unix-like sono chiamate *shared object*. Si tratta del meccanismo che permette di inserire tutto il codice comune usato dai programmi all'interno di opportune librerie, in modo che non sia necessario averne una copia all'interno di ciascun programma. Abbiamo anche visto in sez. 1.2.3 come ci siano delle directory specifiche, previste dal *Filesystem Hierarchy Standard*, che devono contenere queste librerie.

Per far sì che il codice delle librerie possa essere utilizzato dai singoli programmi occorre però una apposita infrastruttura. Come accennato anche in sez. 2.1.5, uno dei compiti fondamentali del sistema, quello di mettere in esecuzione i programmi, viene realizzato attraverso l'ausilio del cosiddetto *link-loader*. Questo è un programma speciale, che non è eseguibile in quanto tale (ed infatti si trova sotto `/lib`), si tratta infatti di una sezione di codice che viene utilizzata direttamente dalla chiamata al sistema che manda in esecuzione un nuovo programma nel momento in cui questo viene caricato in memoria.

Oltre alla gestione degli script accennata in sez. 2.1.5 è compito del *link-loader* caricare opportunamente in memoria i programmi binari,[1] prima di metterli in esecuzione. Il nome deriva dal fatto che oltre a caricare il codice dell'eseguibile il *link-loader* deve anche effettuare il collegamento delle librerie condivise da questo utilizzate.[2] Questo significa che il *link-loader* dovrà identificare le funzioni di libreria utilizzate da un programma, caricarle in memoria ed inserire i relativi indirizzi all'interno del codice, in modo che possano essere invocate.

Per verificare quali librerie dinamiche sono necessarie per l'esecuzione di un programma si può usare il comando `ldd`, che stampa sullo *standard output* i nomi delle librerie condivise di cui esso ha bisogno. Il comando prende come argomento il pathname assoluto del programma da analizzare e stampa a video il risultato, ad esempio:

```
piccardi@hain:~$ ldd /bin/ls
        linux-gate.so.1 =>  (0xb78d6000)
        libselinux.so.1 => /lib/libselinux.so.1 (0xb78b4000)
        librt.so.1 => /lib/i686/cmov/librt.so.1 (0xb78ab000)
        libacl.so.1 => /lib/libacl.so.1 (0xb78a3000)
        libc.so.6 => /lib/i686/cmov/libc.so.6 (0xb775c000)
        libdl.so.2 => /lib/i686/cmov/libdl.so.2 (0xb7758000)
        /lib/ld-linux.so.2 (0xb78d7000)
        libpthread.so.0 => /lib/i686/cmov/libpthread.so.0 (0xb773f000)
        libattr.so.1 => /lib/libattr.so.1 (0xb773a000)
```

ci fa vedere le librerie necessarie all'uso di `ls` (il risultato è relativo alla versione di `ls` installata su una Debian Squeeze su architettura a 32 bit). L'opzione più usata di `ldd` è `-v` che permette di avere una descrizione più dettagliata delle librerie, con le informazioni relative alle versioni, le altre opzioni sono al solito descritte nella pagina di manuale del comando.

Si noti come le librerie siano file il cui nome inizia per `lib` e termina con l'estensione `.so`, che sta appunto per *shared object*, seguita da un numero che è detto *major version*. Questa è una delle caratteristiche più utili in un sistema unix-like: le librerie sono caratterizzate sempre con due numeri di versione, *major* e *minor*, e la convenzione vuole che le interfacce pubbliche delle librerie non debbano mai cambiare fintanto che non cambia la *major version*. Al solito è soltanto una convenzione, anche se quasi universalmente rispettata; ogni tanto qualche programmatore anche non più alle prime armi la viola, con il risultato che programmi che fino ad allora funzionavano perfettamente si trovano a riportare errori o a terminare improvvisamente.

Con questa architettura si ottengono due risultati di grande rilevanza: il primo è che si può cambiare tranquillamente la *minor version* di una libreria senza che i programmi che la usano ne abbiano a risentire, a meno che, al solito, un programmatore non troppo furbo abbia usato una qualche funzione interna che non fa parte della interfaccia pubblica, nel qual caso di nuovo può succedere di tutto. Il secondo è che se si ha bisogno di una versione vecchia delle librerie si può installare anche quella, e sarà tranquillamente usabile attraverso la diversa *major version*, senza nessun conflitto con le versioni più recenti. Questo è il motivo per cui il problema dei conflitti di versione delle librerie, tristemente noto come *DLL hell*, è sostanzialmente assente su GNU/Linux.

[1] in Linux esistono sostanzialmente due formati binari, quello chiamato `a.out`, usato nei kernel delle serie fino alla 1.2, attualmente deprecato e totalmente inutilizzato da anni, e il formato ELF, usato nelle serie successive.

[2] a meno che questo non sia stato compilato staticamente, cioè in maniera da includere al suo interno anche tutto il codice che altrimenti sarebbe stato disponibile attraverso delle librerie condivise, cosa che normalmente non viene mai fatta per ragioni di efficienza.

Dato che il *link-loader* deve essere in grado di determinare quali sono le librerie che contengono le funzioni richieste da un programma tutte le volte che questo viene eseguito, per effettuare la ricerca in maniera efficiente viene utilizzato un apposito file di *cache* in cui sono state indicizzate tutte le informazioni relative alle funzioni presenti ed alle librerie in cui esse sono contenute. Questo file si chiama `/etc/ld.so.cache` e viene generato (di norma tutte le volte che si installa una nuova libreria) con il comando `ldconfig`.

Il comando `ldconfig` permette sia di ricostruire la cache che di ricreare i link all'ultima versione dei file delle librerie; infatti queste vengono sempre cercate per numero di *major version*, ma la libreria installata sarà contenuta in un file con una sua specifica *minor version*, o in generale, come nell'esempio, con un suo specifico numero di versione. Perciò quello che si fa è creare un link simbolico dalla versione cercata alla versione effettiva, e ad esempio avremo che:

```
piccardi@anarres:/lib$ ls -l librt*
-rw-r--r-- 1 root root 30684 Dec 18 00:42 librt-2.11.3.so
lrwxrwxrwx 1 root root    15 Feb  9 12:14 librt.so.1 -> librt-2.11.3.so
```

in cui si può notare come una certa versione specifica di una libreria (la 2.11.3) venga rimappata come prima *major version*.

Un'altra delle funzionalità avanzate di `ldconfig` è che è in grado di gestire in un unico file di cache, per le architetture che le supportano, anche con diversi formati dei binari e delle relative librerie (come avviene per gli eseguibili a 32 bit sulle macchine a 64 bit). Se invocato con l'opzione `-v` il comando stampa l'elenco di tutte le directory esaminate e delle librerie usate nella ricostruzione, mentre con `-N` e `-X` si può bloccare rispettivamente la ricostruzione della cache e quella dei link.

In sez. 1.2.3 abbiamo visto come secondo il *Filesystem Hierarchy Standard* le librerie possono essere mantenute in diverse directory; di default il comando `ldconfig` esamina soltanto le directory `/lib` e `/usr/lib`. Se ci sono altre librerie condivise poste in directory diverse, queste possono essere specificate tramite un opportuno file di configurazione, `/etc/ld.so.conf`. Il file contiene una lista tutte le directory che contengono librerie condivise da far gestire a `ldconfig`, in genere elencate una per riga (ma la lista può essere separata da spazi o da "`:`") e supporta anche, nelle versioni più recenti del comando, l'uso della direttiva `include`, per caricare configurazioni analoghe da altri file. Ad esempio la versione installata su una Debian Squeeze è la seguente:

/etc/ld.so.conf
```
include /etc/ld.so.conf.d/*.conf
```

Se allora si installa una nuova libreria dai sorgenti, questa di norma (vedi sez. 4.2.1) andrà in `/usr/local/lib`, e se questa directory non compare in `/etc/ld.so.conf` o in uno dei file in essa inclusi,[3] sarà necessario aggiungerla in questo file e poi eseguire `ldconfig` per aggiornare i link alle librerie condivise disponibili e ricreare la cache, in modo che il linker dinamico possa utilizzarla.

Il default di `ldconfig` prevede l'uso di `/etc/ld.so.conf`, ma usando l'opzione `-f` si può specificare un qualunque altro file al suo posto, mentre con `-n` si può passare direttamente sulla linea di comando una lista di directory su cui effettuare la ricerca. Per le altre opzioni e la documentazione completa si consulti al solito la pagina di manuale, disponibile con `man ldconfig`.

[3] nel caso di Debian viene dichiarata in `/etc/ld.so.conf.d/libc.conf`, SuSE ad esempio la aggiunge direttamente in questo file.

Infine, nei casi in cui si vogliano utilizzare solo in forma temporanea delle librerie condivise, si può ricorrere alla variabile di ambiente `LD_LIBRARY_PATH`, in cui passare una lista, nella stessa forma usata dalla variabile `PATH` della shell (vedi sez. 2.1.3), di ulteriori directory separate da "`:`", in cui verrà effettuata la ricerca di altre librerie, questa volta senza usare l'indicizzazione, per cui il sistema sarà più lento.

In genere si usa questa variabile quando si sviluppano delle librerie o se si vuole usare qualche pacchetto sperimentale oppure delle versioni alternative delle librerie di sistema. Infatti le librerie contenute nelle directory specificate tramite `LD_LIBRARY_PATH` hanno la precedenza e vengono utilizzate per prime. In questo modo si può fare uso di librerie sperimentali senza conseguenze per gli altri programmi e per l'intero sistema, che continueranno ad usare la versione ordinaria. Si tenga presente infatti che se si installasse una versione non funzionante di una libreria fondamentale come la *glibc*, praticamente tutti i programmi smetterebbero di funzionare.

Un controllo ancora più dettagliato si può avere usando la variabile `LD_PRELOAD`, che prende una lista di file (separati da spazi) di singole librerie che verranno caricate prima di tutte le altre. In questo modo diventa possibile sovrapporsi alle funzioni di una singola libreria senza dover fare riferimento ad una directory alternativa. Per motivi di sicurezza, dato che un attaccante potrebbe utilizzare le proprie funzioni (magari non troppo benevole) al posto di quelle di sistema, quando si usa questa variabile con programmi *suid* o *sgid* (vedi sez. 1.4.2) essa verrà presa in considerazione solo specificando librerie che siano a loro volta *suid* o *sgid* ed installate nelle directory standard `/lib` e `/usr/lib`. Altre variabili di ambiente che controllano l'uso delle librerie dinamiche sono descritte nella pagina di manuale del *link-loader*, accessibile con `man ld.so`.

3.1.3 La gestione dei parametri del kernel con *sysctl*

Una delle funzionalità presenti in tutti i kernel unix-like è la possibilità di modificare alcuni parametri di comportamento del sistema anche una volta che questo sia stato avviato. Tradizionalmente questo veniva effettuato attraverso una opportuna interfaccia a quello che si suole chiamare il sistema del *sysctl* che veniva gestito attraverso delle apposite *system call*.[4]

L'interfaccia prevede che a ciascun parametro che può essere modificato corrisponda una chiave che lo identifica. Queste chiavi vengono opportunamente suddivise in una gerarchia di nomi, in modo da poterle organizzare per categorie e sottocategorie. La singola chiave viene pertanto identificata tramite la sua posizione nella gerarchia, che viene espressa con una sorta di pathname, in cui si prevede che la si indichi con i nomi del percorso che si deve fare a partire dalle categorie iniziali, separati dal carattere "`.`" anche se, per la evidente analogia con un pathname, i comandi usati con Linux prendono come separatore anche il carattere "`/`".

L'impostazione di questi parametri viene effettuata tramite il comando `sysctl`, ed essendo una operazione che riguarda il comportamento del sistema come tale è riservata all'amministratore. Il comando richiede che si specifichi come argomento o il nome di una chiave, nel qual caso ne stamperà il valore, o una assegnazione dello stesso nome ad un valore, nel qual caso assegnerà detto valore al corrispondente parametro del kernel. Questo significa che ad esempio si potrà leggere il valore corrente di un parametro con:

```
anarres:~# sysctl kernel.hostname
kernel.hostname = anarres
```

[4] l'argomento è trattato in dettaglio nella sez. 6.2.1 di [GaPiL].

mentre si potrà assegnarne uno con:

```
anarres:~# sysctl net.ipv4.icmp_echo_ignore_broadcasts=1
net.ipv4.icmp_echo_ignore_broadcasts = 1
```

Il comando accetta anche più argomenti, nel qual caso considererà gli stessi come un elenco, questo comporta che l'assegnazione deve sempre essere eseguita senza spazi fra il nome della chiave, il carattere "=" ed il valore impostato. Il tipo di valore da assegnare ad una chiave dipende dal tipo di parametro che si imposta attraverso di essa, ma i più comuni sono dei numeri interi o delle stringhe. Qualora si debba effettuare una assegnazione con un valore che contiene dei caratteri interpretabili dalla shell questo può essere indicato fra virgolette, ma in tal caso occorre specificare l'opzione `-w`.

Categoria	Significato
`debug`	parametri per il debug.
`dev`	parametri dei dispositivi.
`fs`	parametri dei filesystem.
`kernel`	parametri generici del kernel.
`net`	parametri della rete.
`vm`	parametri della memoria virtuale.

Tabella 3.1: Le principali categorie dei parametri del kernel.

In tab. 3.1 si sono riportate le principali categorie dei parametri del kernel, che stanno alla radice della gerarchia delle chiavi. Un elenco completo di tutte le chiavi disponibili può essere ottenuto invocando `sysctl` con l'opzione `-a`, mentre con l'opzione `-e` si dice al comando di ignorare errori dovuti a chiavi non esistenti. Si può poi usare l'opzione `-p` per far eseguire l'impostazione di una serie di valori leggendo una lista di assegnazioni dal file passato come parametro per l'opzione stessa; se come parametro di `-p` non si specifica nulla viene usato come default `/etc/sysctl.conf`. Al solito per i dettagli e l'elenco completo delle opzioni si consulti la pagina di manuale del comando.

In realtà, come accennato in sez. 1.2.3, in tutte le versioni più recenti di Linux l'uso di un comando dedicato come `sysctl` non è più necessario perché tutta la gerarchia delle chiavi di configurazione viene riproposta direttamente all'interno del filesystem `/proc`, come altrettanti veri e propri pathname a partire da `/proc/sys`. Anzi su Linux l'interfaccia tradizionale, che prevede una apposita *system call*, è stata eliminata nei kernel più recenti, ed anche per poter utilizzare il comando `sysctl` è necessario disporre del filesystem `/proc`. Questo significa che si può effettuare una assegnazione come quella mostrata nel precedente esempio semplicemente con:

```
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

Il vantaggio di poter effettuare le impostazioni operando direttamente sul filesystem `/proc` è che su di esso si possono utilizzare tutti i comandi per la gestione dei file visti in sez. 2.2. Per questo motivo il comando `sysctl` non viene oramai quasi più usato direttamente per effettuare una impostazione, con l'eventuale eccezione della configurazione delle eventuali personalizzazioni dei parametri del kernel da effettuarsi all'avvio del sistema.

A tale scopo infatti viene ancora usato `sysctl`, che viene invocato all'interno della procedura di avvio per eseguire le impostazioni previste dall'amministratore. Queste sono normalmente

mantenute nel file /etc/sysctl.conf cui abbiamo già accennato. Il formato del file prevede la presenza di una assegnazione per riga, nella stessa forma in cui la si scrive sulla riga di comando, con la possibilità di inserire commenti precedendoli con il carattere "#".

Nelle distribuzioni più recenti è stato anche previsto l'uso di una directory /etc/sysctl.d/ all'interno della quale inserire singoli file .conf con impostazioni specifiche in modo da consentire una gestione più flessibile di eventuali personalizzazioni dei parametri del kernel da parte di singoli pacchetti. I file seguono la stessa sintassi di sysctl.conf, ma possono avere un nome qualsiasi. La directory viene analizzata in fase di avvio,[5] e vengono effettuate le impostazioni contenuti nei vari .conf presenti al suo interno, letti in ordine alfabetico. Per ultime vengono sempre applicate le configurazioni di /etc/sysctl.conf.

3.1.4 Localizzazione e internazionalizzazione

Come accennato in sez. 2.1.3 su un sistema GNU/Linux è presente, a partire dalla libreria fondamentale del sistema (la *glibc*), un sistema di supporto per la localizzazione e la internazionalizzazione che consente di personalizzare in maniera del tutto generica il funzionamento dei programmi rispetto alla propria lingua e alle convenzioni riguardo formato di data, ora, valuta, ecc. utilizzate nel proprio paese.

Spesso si trova riferimento a questo supporto, ad esempio nei pacchetti che contengono gli eventuali dati ad esso relativi, con le sigle *l10n* e *i12n* come abbreviazione delle parole inglesi *localization* e *internationalization* (dove i numeri vanno ad indicare il numero di lettere intercorrenti). Non è detto infatti che questo sia sempre disponibile per qualunque lingua, e per alcuni programmi (ad esempio per alcuni ambienti grafici), viste anche le dimensioni ragguardevoli che può assumere, può essere necessaria una installazione separata. Questo non è comunque vero per i comandi e le funzionalità di base (in particolare per la shell ed i vari comandi dei coreutils) per i quali in genere il supporto è sempre compreso nella versione installata.

Il supporto per la localizzazione prevede che si possano marcare all'interno di un programma tutti i messaggi e le stringhe che vengono stampate, così che queste possano essere tradotte e stampante automaticamente nella lingua giusta. Oltre a questo vengono fornite opportune funzioni di interfaccia per modificare l'ordinamento alfabetico e la stampa di alcuni dati, come appunto orari, date e valute, in modo che i programmi si comportino seguendo le convenzioni in uso nel proprio paese.

Questo significa, come già illustrato in sez. 2.1.3, che l'uso della localizzazione non ha impatto soltanto sulla visualizzazione dei messaggi dei comandi o sulle voci dei menù, o sulla lingua in cui vengono mostrate le pagine di manuale, ma anche su aspetti più sottili del comportamento del sistema, come l'ordinamento alfabetico o il significato degli intervalli di caratteri utilizzati per le espressioni regolari e per il *filename globbing*.

Tutte queste funzionalità relative alla localizzazione vengono controllate attraverso l'uso di una serie di variabili di ambiente che sono riportate, insieme al loro significato, in tab. 3.2, maggiori dettagli si possono trovare nella pagina di manuale dedicata all'argomento, accessibile con man 7 locale. Abbiamo già incontrato in sez. 2.1.3 una di queste, LANG, che è quella usata più comunemente in quanto consente di impostare una localizzazione di default valida in generale,

[5] con Debian veniva fatto dallo script di avvio (vedi sez 5.3.5) /etc/init.d/procps, con l'uso di systemd (vedi sez. 5.3.6) come programma di avvio questo viene fatto direttamente da lui.

Variabile	Significato
LANG	imposta il valore generale della localizzazione, che rimane valido per tutti gli aspetti a meno che questi non vengano diversamente specificati tramite una delle variabili LC_*.
LC_COLLATE	modifica il comportamento delle funzioni di confronto fra stringhe, modificando pertanto l'ordinamento alfabetico e la definizione degli intervalli o le corrispondenze nelle espressioni regolari e nel *filename globbing*.
LC_CTYPE	modifica l'identificazione dei caratteri come maiuscoli e minuscoli e l'interpretazione dei caratteri a byte multipli.
LC_MONETARY	modifica le modalità con cui vengono stampati i valori numerici che esprimono valori monetari, in particolare l'uso della virgola o del punto per la definizione dei decimali e la stampa del tipo di valuta.
LC_MESSAGES	modifica la lingua in cui vengono mostrati i messaggi dei programmi, ed anche la forma delle risposte positive e negative.
LC_NUMERIC	modifica le modalità con cui vengono letti e stampati i numeri, in particolare per l'uso del punto o della virgola per i decimali.
LC_TIME	modifica la modalità con cui vengono stampati data ed ora.
LC_ALL	imposta il valore per tutte le variabili precedenti, sostituendo eventuali impostazioni eseguite tramite esse.
LANGUAGE	qualora si sia impostato un valore di LANG che indica una localizzazione diversa dal default, si può usare questa variabile per indicare una lista di priorità delle localizzazioni da usare se quella espressa da LANG non è disponibile, in forma di elenco separato dal carattere ":" (ad esempio il valore it:fr mostra il francese se questo è disponibile quando non lo è l'italiano).

Tabella 3.2: Le variabili di ambiente che controllano il comportamento della localizzazione.

senza dover impostare i singoli aspetti della localizzazione impostando le singole variabili LC_* una per una.

La variabile LANG e le varie LC_* vengono normalmente inizializzate a seconda della lingua in cui si è installato il sistema,[6] ma si può verificarne il valore con il comando `locale` che ne stampa a video l'impostazione corrente. La localizzazione di default presente nel codice della *glibc* e dei comandi di base (sostanzialmente l'inglese) usa il valore riservato C; alternativamente, per indicare esplicitamente l'aderenza allo standard, si può utilizzare il valore POSIX.

È possibile comunque installare anche delle localizzazioni ulteriori, e passare dall'una all'altra, anche per un singolo aspetto, semplicemente ridefinendo i valori delle variabili di tab. 3.2. Ad esempio si può usare la localizzazione italiana con il valore it_IT.UTF-8. Questo già ci mostra il formato generico dei valori utilizzabili per queste variabili, che riportano in minuscolo il valore principale della lingua (nel caso it) seguito da un carattere di sottolineatura e dal valore secondario in maiuscolo (nel caso dell'Italia è di nuovo IT, ma per la Svizzera italiana sarebbe stato CH), cui segue, separato da un punto, il nome della codifica dei caratteri (che nell'esempio è UTF-8, ma sarebbe potuto essere anche qualcosa come ISO-8859-1).

La lista di questi valori viene in genere mantenuta nel file /etc/locale.alias, che serve ad avere anche una corrispondenza con dei nomi più comprensibili per l'utente, il file ha il semplice formato di un nome per esteso (ad esempio Italiano) seguito dal valore di localizzazione corrispondente, separato da spazi. Vari servizi, in particolare quelli che danno accesso all'interfaccia

[6]le modalità con cui questo viene realizzato variano da distribuzione a distribuzione, Debian ad esempio usa il valore contenuto nel file /etc/default/locale negli script di avvio.

grafica, fanno riferimento a questo file (o a opportune copie dello stesso), per presentare menù di scelta della lingua.

Ovviamente perché sia possibile utilizzare una diversa localizzazione occorre che i relativi dati siano installati. Questo a seconda della distribuzione può comportare l'installazione di pacchetti aggiuntivi; inoltre dato che in genere non serve disporre di tutte le localizzazioni possibili, normalmente solo un sottoinsieme fra quelle esistenti (visualizzabile con l'opzione `-a` di `locale`) viene reso disponibile e può essere utilizzato.

In genere infatti oltre alla installazione dei pacchetti con i dati, occorre anche generare, in particolar modo per l'uso delle traduzioni, i file con gli indici che rendono possibile un uso veloce dei messaggi tradotti. Tutto questo comporta ovviamente l'uso di spazio disco ulteriore, che può essere risparmiato quando non necessario. In tal caso esistono in genere gli opportuni strumenti per aggiungere o rimuovere le singole localizzazioni dall'elenco di quelle utilizzabili. Il comando per effettuare questa operazione è `locale-gen`, ma le distribuzioni utilizzano in genere programmi di più alto livello, quando addirittura non rendono utilizzabili di default tutte le localizzazioni disponibili.

Si tenga presente inoltre che oltre a lingua e paese a cui fa riferimento, ciascuna localizzazione è legata anche alla codifica di carattere utilizzata. In generale infatti, dovendosi poter esprimere le stringhe in tutte le possibili lingue esistenti, l'uso degli ordinari caratteri ASCII che originariamente utilizzavano soltanto 7 bit non è sufficiente, in quanto questi possono esprimere un insieme molto limitato di lettere e numeri (in sostanza quelli in uso nei paesi anglosassoni). Già per poter utilizzare i caratteri in uso nelle principali lingue europee infatti sono necessarie delle estensioni rispetto all'ASCII standard, che aggiungano caratteri ulteriori come le vocali accentate, il che comporta l'uso di una serie di codifiche più specifiche come appunto la citata `ISO-8859-1`, definite sulla base di una opportuna standardizzazione.[7]

Ma anche così, mantenendo la convenzione per cui ciascun carattere viene rappresentato da un numero ad 8 bit, si riescono a coprire sostanzialmente soltanto i caratteri in uso nelle lingue occidentali, tanto che la codifica `ISO-8859-1` viene chiamata *Western European*, o anche `latin-1`, dato che per le lingue latine dell'Europa centro-orientale viene usata la codifica `ISO-8859-2`, detta anche `latin-2` o *Central European*.

Per questo già lo standard ISO 8859 prevede numerose codifiche diverse, usate oltre che per le varie varianti dei caratteri latini, anche per consentire l'uso di caratteri cirillici o greci, o per l'aggiornamento delle codifiche con il simbolo dell'euro, non ancora presente all'epoca della definizione dell'`ISO-8859-1` e affini. In genere tutte queste codifiche vengono indicate con il nome `ISO-8859-N`, dove "`N`" è un numero che identifica la singola versione.

Ovviamente questa proliferazione di codifiche diverse comporta una notevole complessità dovendo ad esempio fornire traduzioni con codifiche dei caratteri sempre diverse e l'impossibilità di rappresentare stringhe in cui siano presenti caratteri di lingue diverse. Per questo motivo a partire dal 1991 è stato sviluppato un ulteriore standard, denominato *Unicode*,[8] che consentisse di rappresentare in un'unica codifica qualunque tipo di caratteri, rinunciando però alla pretesa di utilizzare un solo byte per carattere.

[7] nel caso si fa riferimento appunto allo standard ISO/IEC 8859, ma non tutti hanno aderito da subito allo stesso, e ad esempio un noto produttore di sistemi operativi abitualmente refrattario all'uso degli standard ha sviluppato anche codifiche alternative come quella denominata `Windows-1252`, che hanno avuto una certa diffusione.

[8] per i fan delle numerazioni lo standard è classificato ufficialmente come ISO/IEC 10646.

Per far questo vengono utilizzate opportune sequenze di valori numerici, e la forma più utilizzata è quella del citato `UTF-8`, che prevede codici composti da uno a quattro byte per singolo carattere mantenendo una compatibilità, per i caratteri corrispondenti, con il vecchio ASCII. Esistono però, ancorché usate in maniera inferiore, codifiche alternative come `UTF-16` che usa codici composti da una a due parole a 16 bit, o `UTF-32` che usa direttamente parole a 32 bit. Ad oggi, vista la minore complessità di gestione, la gran parte delle distribuzioni tende comunque ad allinearsi all'uso esclusivo di `UTF-8`.

Se dal punto di vista del funzionamento dei programmi l'uso della localizzazione nelle diverse codifiche di caratteri non comporta, una volta installato il relativo supporto, nessun problema, con i dati, ed in particolare con i file di testo, i problemi invece sussistono. Questi infatti possono essere creati su un sistema che usa una certa codifica, e riletti su uno che ne usa un'altra, con risultati in genere spiacevoli dato che si troveranno quelli che prima erano normali caratteri trasformati in simboli strani o in sequenze numeriche di scarso significato.

Per questo motivo è disponibile un apposito programma, `iconv`, che consente di effettuare conversioni di un file di testo da una codifica di caratteri ad un'altra. Il comando richiede che si specifichi come parametro per l'opzione `-f` la codifica del file da convertire, che deve essere passato come argomento, la conversione verrà scritta sullo *standard output*, o sul file indicato con l'opzione `-o`. Se non si specifica, con l'opzione `-t`, una codifica di destinazione, verrà usata quella corrispondente alla localizzazione in uso.

Per indicare una codifica si deve utilizzare il relativo nome, che è lo stesso che poi si usa per le variabili di tab. 3.2, ma dato che il comando supporta le conversioni da e verso ad una grande quantità di codifiche diverse, si può ottenere la lista completa di tutti i relativi identificativi con l'opzione `-l`. Al solito per i dettagli sul funzionamento si consulti la pagina di manuale.

Un ultimo aspetto delle problematiche di localizzazione ed internazionalizzazione è quello dell'uso dell'ora locale. Benché affine alle altre problematiche appena trattate, la gestione di questo aspetto è completamente separata, dato che si può essere all'estero, ed usare un fuso orario diverso da quello del proprio paese d'origine, senza però voler usare la lingua del posto dove ci si trova, o anche solo aver cambiato fuso, se si vive in un paese abbastanza grande da averne più di uno.

Come illustrato in sez. 2.4.3 il kernel opera facendo riferimento al tempo universale.[9] Per poter tener conto del proprio fuso orario, ed anche della eventuale presenza dell'ora locale, esiste un apposito supporto, anche questo integrato nelle librerie fondamentali di sistema, che consente di effettuare la conversione dal tempo universale all'ora locale in maniera trasparente per l'utente. Questo consente di far sì che per il kernel il tempo sia sempre crescente in maniera costante, come è nella realtà, e di evitare salti o buchi nell'orologio. Così se si è salvato un file da tre ore questo avrà sempre come data di ultima modifica tre ore fa, anche se nel frattempo ci si è spostati in un paese diverso, si è cambiato fuso, o è scattata l'ora legale.

Il meccanismo si basa sul concetto di *timezone*, una sorta di estensione del concetto di fuso orario, che consente di incorporare anche le eventuali variazioni dovute alle presenze di regolamentazioni locali (come l'ora legale) che possono variare da paese a paese, e di anno in anno. Il database viene mantenuto a livello globale tenendo conto delle varie regolamentazioni in vigore nei vari paesi, e dato che ci sono variazioni amministrative all'interno dei singoli paesi,

[9] a meno di non aver forzato in fase di installazione l'uso dell'ora locale, per compatibilità con sistemi operativi mal progettati che han fatto questa scelta; questa è una opzione ormai quasi completamente scomparsa e comunque da evitare perché porta ad un uso non corretto del sistema: pertanto la ignoreremo totalmente.

le *timezone* sono in genere indicate con una coppia di nomi del tipo `Area/Locazione`, dove per l'area si fa riferimento ad un continente o ad un oceano (con valori come `Europe`, `Asia`, `Pacific`, ecc.), mentre la locazione fa riferimento o a una città o a una isola, (con valori come `Rome`, `New_York`, `Samoa`, ecc.) considerata come il riferimento per quella zona.

La scelta di evitare l'uso del nome delle nazioni è voluta, sia perché ci sono nazioni con più di un fuso orario, come gli Stati Uniti o la Russia, sia perché l'uso dei nomi delle città principali si è rivelato più stabile e meno equivoco rispetto ai cambiamenti geopolitici. Nel caso dell'Italia ad esempio si usa `Europe/Rome`, in cui, come per buona parte dei paesi europei, si fa riferimento alla capitale.

Ciascuna *timezone* è descritta da un apposito file, chiamato in gergo *zonefile*, che contiene tutte le informazioni ad essa relative che consentono di effettuare la conversione dal tempo universale. Si tratta di uno specifico database (con un formato proprio, per approfondimenti si consulti `http://en.wikipedia.org/wiki/Zoneinfo`) che contiene anche uno storico dei dati per permettere di effettuare la conversione non solo per il tempo corrente, ma anche per tutti i tempi passati.

Il *Filesystem Hierarchy Standard* illustrato in sez. 1.2.3 prevede che questi file vengano installati sotto la directory `/usr/share/zoneinfo`, in altrettanti file e directory corrispondenti al nome della *timezone* stessa, che può essere considerato come un pathname relativo rispetto a `/usr/share/zoneinfo`. La *timezone* in uso viene invece impostata tramite il file `/etc/localtime`, che altro non è che la copia del file della zona scelta preso da `/usr/share/zoneinfo`. Questo, nonostante contravvenga la regola di non usare file binari all'interno di `/etc`, è dovuto al fatto che la relativa informazione deve essere disponibile all'avvio, quando non è detto che `/usr` sia stata montata, ed è per questo che anche l'uso di un link simbolico non è una soluzione adeguata.

Uno dei problemi che si riscontrano nella gestione dei fusi orari è che ciascuna distribuzione usa un suo metodo per impostare la zona di default in `/etc/localtime`,[10] in modo da essere in grado di gestirne l'aggiornamento quando viene installata in `/usr/share/zoneinfo` una nuova versione della relativa *timezone*. Purtroppo questo non è stato standardizzato e modificare a mano questo file può dar luogo ad inconvenienti.

Qualora si voglia usare una zona diversa si dovrà anzitutto determinare l'identificativo ad essa associato, per semplificare l'operazione si può usare il comando `tzselect` che con una serie di domande richiede prima il continente e poi il proprio paese, ed infine stampa il valore dell'ora locale nella zona risultante dalle proprie scelte. Una volta che si sia confermata la correttezza dell'ora ottenuta il comando stampa il nome della relativa zona ed alcune istruzioni. In generale se si vuole effettuare un cambiamento permanente a livello di sistema è opportuno usare il meccanismo che ciascuna distribuzione mette a disposizione, perché anche se si può sempre copiare a mano il file da `/usr/share/zoneinfo`, questo potrebbe venire sovrascritto da un aggiornamento o anche da un riavvio.

Il cambiamento della *timezone* a livello di sistema richiede ovviamente i privilegi di amministratore, ma il singolo utente può effettuare un cambiamento locale, che coinvolga solo lui, o anche singole applicazioni, tramite la variabile di ambiente `TZ`, assegnando ad essa il nome della *timezone* che vuole utilizzare, in tal caso le applicazioni faranno riferimento a quest'ultima invece che a quella di sistema.

[10] su Debian è ancora presente il vecchio comando `tzconfig`, ma questo è deprecato in favore della riconfigurazione del pacchetto dei dati delle *timezone* con (vedi sez. 4.2.3) `dpkg-reconfigure tzdata`, su RedHat occorre invece usare il comando dedicato `system-config-date`.

3.2 I servizi di base

In questa sezione prenderemo in esame la configurazione di alcuni servizi di base, presenti fin dalle origini in tutti i sistemi unix-like, come quelli per l'esecuzione periodica o differita dei comandi, e quelli per la gestione della registrazione dei messaggi di servizio. Seguendo la filosofia fondamentale di Unix tutti questi servizi sono realizzati tramite degli appositi programmi, detti *demoni* (si ricordi quanto detto in sez. 2.1.2), che lavorano in background anche quando nessuno è collegato al sistema ed il cui comportamento viene controllato attraverso i rispettivi file di configurazione.

3.2.1 I servizio di esecuzione periodica

Una funzionalità importante presente in qualunque sistema operativo è quella dell'esecuzione di programmi su base periodica, essenziale per compiere tutte quelle operazioni che devono essere eseguite ad un tempo fissato. Abbiamo già incontrato due operazioni che vengono gestite in questo modo, come la creazione del database dei file di `locate` e l'indicizzazione delle pagine di manuale, trattate rispettivamente in sez. 2.2.2 e 2.4.2.

La gestione dell'esecuzione periodica di programmi per compiere operazioni nel sistema viene realizzata dal servizio chiamato *cron*, che è stato implementato attraverso l'uso dell'omonimo demone `cron`. Il demone ha il compito di svegliarsi ogni minuto ed eseguire ogni programma che è stato programmato per quel momento.

Il file di configurazione principale di `cron` è `/etc/crontab` che contiene l'elenco delle operazioni periodiche generali da eseguire nel sistema. Il primo file controllato da `cron` per decidere se c'è da eseguire una operazione è questo. In genere si deve intervenire su questo file solo quando si vuole cambiare uno degli orari a cui le operazioni prestabilite vengono eseguite o per inserire una nuova operazione periodica. Oltre a `/etc/crontab` il programma controlla anche il contenuto della directory `/etc/cron.d/` dove i vari pacchetti possono inserire i loro file per esigenze specifiche, per i quali vale la stessa sintassi di `/etc/crontab`.

Il formato del file segue la solita regola di ignorare righe vuote ed inizianti per "`#`". Ogni riga deve contenere o una assegnazione di una variabile di ambiente o la specificazione di una azione periodica. L'azione viene specificata da una serie di campi separati da spazi o tabulatori, i primi cinque indicano la periodicità con cui il comando indicato nell'ultimo campo viene eseguito, il sesto campo indica l'utente per conto del quale eseguire il comando, che viene scritto nel seguito della riga.

I cinque campi della periodicità indicano rispettivamente: minuto (da 0 a 59), ora (da 0 a 23), giorno del mese (da 1 a 31), mese dell'anno (da 1 a 12) e giorno della settimana (da 0 a 7, dove sia 0 che 7 indicano la domenica). Per quest'ultimo campo sono accettati anche valori tipo Mon, Thu, etc. L'utilizzo del carattere "*" vale da jolly e lo si usa per indicare un valore qualsiasi. Se il tempo corrente corrisponde a tutti i valori specificati nei cinque campi il comando viene eseguito.

Il demone `cron` di GNU/Linux supporta poi alcune estensioni non presenti in altri sistemi unix-like.[11] Si può usare una lista (separata da virgole) per indicare più valori, un intervallo, specificando gli estremi separati con un "-". Infine il programma supporta la possibilità di

[11]si tratta del programma *vixie-cron*, creato da Paul Vixie, noto anche per essere l'autore di *BIND*, il server DNS (vedi sez. 9) più usato al mondo.

specificare periodi personalizzati attraverso l'uso del carattere jolly abbinato con il carattere "/" seguito dal divisore da applicare al valore generico del campo; così ad esempio usando */2 nel primo campo si intenderà chiedere la ripetizione del lavoro ogni due minuti. Per la descrizione del formato del file completa di tutti i dettagli, si faccia riferimento alla pagina di manuale del file, accessibile con `man 5 crontab`.

Si tenga presente che siccome il programma specificato nella parte conclusiva della linea viene eseguito direttamente dal demone `cron`, non saranno necessariamente definite le usuali variabili di ambiente presenti normalmente quando si esegue una shell. In particolare è da tenere presente che il demone si limita a definire solo alcune di esse ed in particolare per quanto riguarda `PATH` questa comprenderà soltanto le directory `/usr/bin` e `/bin`. Altre variabili definite da `cron` sono `LOGNAME` e `HOME`, che corrisponderanno al nome utente e alla directory personale dell'utente per conto del quale viene eseguito il programma.

Per questo motivo se un programma necessita della presenza di valori diversi da quelli di default, o di altre variabili, occorrerà inserire in testa al file le relative definizioni. Queste definizioni seguono la stessa sintassi che si usa nella shell, con la differenza che `cron` non esegue la *variable expansion*, per cui non si potrà fare riferimento ad altre variabili all'interno di una definizione.

Infine un ruolo particolare viene giocato dalla variabile `MAILTO`, che indica l'utente a cui inviare per posta elettronica l'eventuale output del programma posto in esecuzione. Infatti quando `cron` esegue il comando questo non è collegato a nessun terminale, e si pone pertanto il problema di come gestire gli eventuali dati in uscita. La soluzione scelta è stata appunto quella di inviarli per posta elettronica all'utente per conto del quale è stato eseguito il comando. Questo a meno che non si sia definita la variabile `MAILTO` per indicare un altro destinatario, inoltre se la variabile è definita, ma vuota, l'invio verrà disabilitato. Questo è il motivo per cui spesso nei comandi lanciati tramite `cron` viene eseguita una redirezione dello *standard output* su `/dev/null`, in modo da scartare eventuali dati in uscita.

Un esempio del contenuto del file `/etc/crontab`, preso dalla versione installata di default su una Debian Squeeze, è il seguente:

/etc/crontab

```
# Unlike any other crontab you don't have to run the 'crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user        command
17 *  * * * root cd / && run-parts --report /etc/cron.hourly
25 6  * * * root test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6  * * 7 root test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6  1 * * root test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
```

in cui vengono definite le variabili che indicano quale shell usare ed il path dei comandi e vengono impostate una serie di operazioni. In questo caso ad esempio la seconda riga si legge come minuto

25, ore 6, essendo le altre indicazioni tutti * si intende che giorno del mese, mese e giorno della settimana sono qualsiasi, allo stesso modo si interpretano le altre righe.

Con il contenuto mostrato nell'esempio ci saranno allora delle azioni che vengono eseguite ogni ora al minuto 17, tutti i giorni alle ore 6:25, tutte le domeniche alle 6:47 e tutti i primi giorni del mese alle 6:52, che vanno a costituire le operazioni orarie, giornaliere, settimanali e mensili da eseguire in maniera automatica. Queste azioni sono eseguite attraverso il comando `run-parts` il cui scopo è effettuare l'esecuzione, in ordine alfabetico, degli script presenti nella directory passata come argomento.

Nel caso gli script sono quelli delle directory elencate nell'esempio, per cui se non si hanno esigenze particolari non è il caso di intervenire su questo file ma di aggiungere il proprio script direttamente in `/etc/cron.daily/`, `/etc/cron.weekly/`, `/etc/cron.monthly/`. Occorre comunque fare attenzione, perché `run-parts` non esegue gli script il cui nome contiene caratteri diversi da lettere maiuscole e minuscole, numeri, il carattere di sottolineatura ed il "`-`" (per i dettagli sul funzionamento del comando si consulti la relativa pagina di manuale) per cui è meglio evitare di installare un `qualcosa.sh`.

Si noti comunque come nel file di `crontab` mostrato nell'esempio viene fatto un controllo preventivo della presenza del comando `anacron`. Questo è un sostituto di `cron` utilizzato originariamente sulle macchine che non è previsto siano sempre accese, come le stazioni di lavoro, che si incarica di eseguire le operazioni periodiche tenendo conto anche del fatto che queste possono non essere state effettuate in precedenza perché la macchina era spenta.

In Debian se questo programma viene installato è quello a cui viene delegato il compito anche di eseguire le operazioni giornaliere, settimanali e mensili illustrate in precedenza e non è necessario l'intervento di `cron`. L'installazione di default non prevede la presenza di questo programma, per cui il comando non sarà presente e verrà eseguito invece `run-parts`.

Altre distribuzioni come RedHat installano invece di default questo programma e delegano direttamente ad esso l'esecuzione delle operazioni periodiche. Queste sono controllate dal file `/etc/anacrontab` che prevede di nuovo la possibilità di definire delle variabili di ambiente nella forma di una assegnazione della stessa come nella shell. Al solito le righe vuote e tutto quello che segue un "`#`" viene ignorato, mentre le righe restanti servono ad indicare le operazioni da eseguire.

Ciascuna operazione deve essere specificata in una riga con 4 campi separati da spazi, che indicano rispettivamente: una periodicità di esecuzione (in numero di giorni), un ritardo iniziale (in numero di minuti), una stringa di identificazione, ed il comando da eseguire. Il primo campo, che indica la periodicità, può essere anche espresso in forma simbolica con i nomi `monthly`, `weekly` e `daily` preceduti dal carattere "`@`". Un esempio di questo file, sempre preso dalla versione installata su Debian Squeeze, è il seguente:

/etc/anacrontab

```
# /etc/anacrontab: configuration file for anacron

# See anacron(8) and anacrontab(5) for details.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# These replace cron's entries
1       5       cron.daily      nice run-parts --report /etc/cron.daily
```

```
7        10      cron.weekly      nice run-parts --report /etc/cron.weekly
@monthly 15      cron.monthly     nice run-parts --report /etc/cron.monthly
```

Il funzionamento di `anacron` prevede che il comando legga la lista delle operazioni periodiche da `/etc/anacrontab` e controlli se queste sono state eseguite nel periodo di tempo indicato nel primo campo. Il programma infatti mantiene una sua lista interna delle operazioni, in cui registra anche l'ultima esecuzione effettuata, e se è passato più tempo di quello richiesto mette in esecuzione il relativo comando dopo il ritardo indicato dal secondo campo, che viene utilizzato per evitare di lanciare in contemporanea tutti i comandi relativi alle operazioni scadute.

Il programma legge il file all'avvio e poi resta in esecuzione come demone, ripetendo le operazioni secondo la periodicità indicata, che però ha come granularità minima quella dei giorni (viene cioè controllata solo la data e non l'ora). Questo, fintanto che la macchina è accesa, garantisce la periodicità nei termini indicati, ma consente, se ad esempio si spegne la macchina la notte, di eseguire i compiti rimasti sospesi al riavvio successivo non essendo vincolato, come `cron`, ad un orario specifico.

Si tenga presente però che a parte gestire le operazioni periodiche in questa modalità, `anacron` non costituisce una alternativa a `cron`, sia per l'impossibilità di usare periodicità su tempi inferiori al giorno sia per l'impossibilità di indicare orari precisi di esecuzione, che sono necessari in molti casi. Al solito per i dettagli sul funzionamento e sulla configurazione, si consultino le relative pagine di manuale con `man anacron` e `man anacrontab`.

Oltre ai compiti relativi alla gestione del sistema, e programmati pertanto dall'amministratore che è l'unico che ha i privilegi per modificare `/etc/crontab`, il servizio *cron* è in grado di eseguire anche operazioni richieste da un singolo utente. L'utente può creare un suo *crontab* personale tramite il comando `crontab`, in questo caso i file vengono mantenuti in `/var/spool/cron/crontabs`, e non sono accessibili direttamente ai singoli utenti se non appunto attraverso detto comando.

Se si invoca `crontab` con l'opzione `-e` questo invocherà l'editor predefinito con cui scrivere il contenuto del proprio *crontab* personale. Il default è l'uso di `vi`, a meno di non aver impostato diversamente la variabile di ambiente `EDITOR`. Il formato delle tabelle di *cron* personali è identico a quello di `/etc/crontab`, con l'eccezione dell'assenza del sesto campo, che nel caso non è necessario dato che l'utente è già ben definito.

Se lanciato senza opzioni `crontab` prende come argomento un file da usare come tabella, di cui legge il contenuto, che deve essere nella forma in cui lo si scriverebbe con l'editor, invece usando "`-`" come nome del file i dati verranno presi dallo *standard input*. Con `-r` invece si esegue la rimozione completa della tabella corrente, ma si faccia attenzione nell'uso dato che il comando non chiede conferma. Infine con `-u` si potrà modificare (solo da parte dell'amministratore) la tabella dell'utente specificato come parametro.

Si può poi vedere la lista delle operazioni programmate usando l'opzione `-l`, che scrive sullo *standard output* il contenuto corrente della tabella *cron* dell'utente. La versione usata da Debian in realtà non stampa le righe di commento iniziale che avvisano di non modificare direttamente il contenuto del *crontab*, in modo da poter riutilizzare l'output del comando direttamente come input per un successivo `crontab -`.

Si tenga presente che l'amministratore può riservare l'uso del servizio *cron* solo ad alcuni utenti, creando il file `/etc/cron.allow`, nel quale si devono elencare gli username degli utenti che lo possono usare. Se il file non esiste è possibile vietare l'uso del servizio solo ad alcuni utenti

con la creazione del file `/etc/cron.deny` che parimenti dovrà contenere la lista degli username di coloro che non possono usarlo. In assenza di detti file chiunque può usare il servizio.

3.2.2 Il servizio *at*

Abbiamo visto nella sezione precedente come si può programmare l'esecuzione periodica di comandi con `crontab`, esistono però anche necessità di avere una esecuzione semplicemente differita, senza che questa debba essere periodica. Per provvedere a questo compito esiste un altro servizio, chiamato *at* dal nome del comando che permette di richiedere l'esecuzione di un programma ad un tempo successivo. Il servizio comporta diverse modalità di gestione della esecuzione differita, e vari programmi per la gestione della stessa.

Il servizio è realizzato dal demone `atd`, che come `cron` viene normalmente lanciato dagli script di avvio. Il demone controlla la presenza nelle code di programmi in attesa di essere eseguiti, e li manda in esecuzione al tempo stabilito. Il demone può essere anche lanciato manualmente, e prevede come opzioni `-b` che permette di impostare un intervallo di tempo minimo (il default è 60 secondi) fra la messa in esecuzione di due successivi programmi e `-l` che permette di impostare manualmente un limite massimo per il carico della macchina, oltre il quale non saranno comunque eseguiti i programmi, di default questo è pari a 1.5, ed ha senso cambiarlo solo in caso di macchine multiprocessore. Al solito per l'elenco completo delle opzioni si può fare riferimento alla pagina di manuale.

Il comando di base, che permette di programmare l'esecuzione di un programma ad un dato momento, è `at`. Il comando vuole come argomento data ed ora in cui il programma specificato deve essere messo in esecuzione; il nome di quest'ultimo viene letto dallo *standard input* ma può essere specificato tramite un file qualunque, passato come parametro per l'opzione `-f`. Le altre principali opzioni sono riportate in tab. 3.3, per l'elenco completo si può al solito consultare la pagina di manuale.

Opzione	**Significato**
`-f`	indica il file da cui leggere il comando da eseguire.
`-l`	stampa la lista dei lavori programmati (equivalente a `atq`).
`-d` *`job`*	rimuove il lavoro con *job id* `job` (equivalente a `atrm job`).
`-q` *`queue`*	invia il comando sulla coda `queue`.[12]
`-m`	invia un messaggio al completamento del lavoro programmato anche nel caso non ci siano dati in uscita.

Tabella 3.3: Principali opzioni del comando `at`.

Il comando supporta una grande varietà di formati per le modalità in cui si indica la data, fra cui tutti quelli del comando `date` (vedi sez. 2.4.3), più altre estensioni che permettono di usare anche specificazioni in forma più *discorsiva* come `at 1pm tomorrow`; una descrizione completa delle convenzioni utilizzabili si trova nel file `/usr/share/doc/at/timespec`. Una volta completata la programmazione il programma stampa sul terminale un valore numerico (detto *job id*) che identifica il lavoro differito programmato dall'utente. Un esempio dell'uso di `at` è il seguente:

[12] il servizio *at* prevede l'esistenza di diverse code con diverse priorità, queste sono identificate dalle lettere dell'alfabeto dall'`a` alla `z`; il default prevede che il comando `at` usi la coda "`a`" ed il comando `batch` la coda "`b`", le altre code devono essere indicate esplicitamente ed i comandi posti in esse verranno eseguiti con un più alto valore di *nice* (cioè con priorità inferiore).

```
piccardi@anarres:~$ at 18:00
warning: commands will be executed using /bin/sh
at> ps aux > ~/listaprocessi.txt
at> <EOT>
job 1 at Wed Mar 19 18:00:00 2008
```

Si tenga presente che la directory di lavoro, la maschera dei permessi e l'ambiente del comando che verrà lanciato sono quelli presenti al momento in cui si invoca `at`, inoltre il comando sarà eseguito usando `/bin/sh` come shell. Come per *cron* in caso di presenza di dati in uscita (sia sullo *standard output* che sullo *standard error*) questi verranno inviati all'utente che ha programmato l'esecuzione con un messaggio di posta elettronica.

Una alternativa ad `at` per l'esecuzione differita è l'uso del comando `batch` (che in realtà è un equivalente di `at -b`) che permette di programmare l'esecuzione di un programma non ad un orario predefinito, ma in base al carico della macchina. Il comando richiesto cioè verrà posto in esecuzione soltanto quando il carico medio della macchina scende al di sotto di un certo valore.

Una volta programmata l'esecuzione di un programma (sia con `at` che con `batch`) questo viene messo, per ciascun utente, in una opportuna coda. Gli utenti possono vedere la lista dei propri programmi in attesa di esecuzione con il comando `atq`, mentre l'amministratore può usare lo stesso comando per vedere la lista di tutti quanti. Per ciascun programma `atq` stampa il valore del *job id* ad esso associato e l'orario per cui è stata programmata l'esecuzione, ad esempio:

```
piccardi@anarres:~$ atq
1       Wed Mar 19 18:00:00 2008 a piccardi
```

Un programma può essere rimosso dalla coda con il comando `atrm`, che prende come argomento il valore del *job id* ad esso associato. Di nuovo un utente normale può operare solo sui propri programmi, mentre l'amministratore può operare su tutti quanti. Infine, come per l'uso del servizio *cron*, è previsto un controllo degli accessi al servizio *at* attraverso i due file `/etc/at.allow` e `/etc/at.deny`, il cui formato e significato sono identici a quelli degli analoghi `cron.allow` e `cron.deny`.

3.2.3 Il servizio del *syslog*

Il *syslog* è il servizio usato dai demoni che girano in background e dal kernel stesso quando devono registrare dei messaggi, in inglese infatti si parla di *log* quando si fa riferimento alla registrazione degli eventi, come per il giornale di bordo di una nave, o il libro su cui si annotano le attività giornaliere; il *syslog* serve appunto a tenere il giornale di bordo di quanto avviene nel vostro sistema.

Come già illustrato in sez. 1.3.2 i demoni devono essere attivi anche quando nessuno è collegato al sistema, e per questo non sono associati ad un terminale, non è quindi loro possibile scrivere dei messaggi di avviso o di errore sullo *standard output* o lo *standard error*, dato che nessuno li leggerebbe. Per questo motivo viene fornito (dalla libreria di sistema, la *glibc*) un servizio di registrazione a cui i vari programmi che hanno qualcosa da riportare possono lasciare i loro messaggi.

Il servizio viene chiamato genericamente *syslog*, e la sua gestione tradizionalmente veniva effettuata dal programma `syslogd` che si incaricava di raccogliere questi messaggi e registrarli in maniera opportuna. Oggi quasi tutte le distribuzioni più importanti (Debian, RedHat, SuSE)

sono passate all'uso di `rsyslogd`, una reimplementazione compatibile con il vecchio `syslogd` che offre una serie di estensioni ed un maggior controllo e rimuove la necessità di dover usare un secondo programma (`klogd`) per raccogliere anche i messaggi del kernel.[13]

Dato che le sue funzionalità sono utilizzate dalla gran parte degli altri programmi di servizio, in genere il *syslog* viene attivato nelle prime fasi della procedura di avvio del sistema (tratteremo l'argomento in sez. 5.3.5). In caso di necessità si può comunque lanciare direttamente `rsyslogd` dalla riga di comando, ad esempio se per un qualche motivo il servizio non parte e si vuole capire cosa succede utilizzando l'opzione `-d` che lo avvia in modalità di debug, in cui resta agganciato al terminale e stampa una serie di informazioni relative al suo funzionamento.

Nella configurazione ordinaria `rsyslogd` riceve i messaggi inviati localmente dagli altri programmi, (per i quali viene usato un socket dedicato, `/dev/log`) ma è possibile abilitare la ricezione di messaggi anche via rete da altre macchine. Per il vecchio `syslogd` questo veniva fatto lanciandolo con l'opzione `-r`, per `rsyslogd` esiste una apposita direttiva di configurazione, mentre la principale opzione a riga di comando è `-c` che indica il livello di compatibilità con `syslogd` da adottare, e che se non indicata implica il livello massimo.

Il vantaggio di `rsyslogd` è che il programma è stato sviluppato in maniera modulare, ed è possibile attivare funzionalità aggiuntive, come quella di gestire direttamente i log del kernel, o di fornire il servizio di *syslog* via rete, attivando i relativi moduli. La descrizione completa del comando e di tutte le opzioni è disponibile nella pagina di manuale, al solito accessibile con `man rsyslogd`.

Il file di configurazione che consente di controllare cosa viene registrato ed in quale modalità è `/etc/rsyslog.conf` e supporta la stessa sintassi usata per il file `/etc/syslog.conf` che veniva usato con `syslogd`, tanto che è possibile semplicemente copiare quest'ultimo per riottenere lo stesso comportamento. Entrambi i file seguono le convenzioni comuni, per cui le linee vuote o che iniziano per "`#`" vengono ignorate. Tratteremo più avanti le estensioni specifiche di `/etc/rsyslog.conf`, che in genere si trovano in testa a questo file, esaminando per prima la sintassi delle regole utilizzate per la registrazione (che sono l'unico contenuto di `/etc/syslog.conf`) che resta la stessa.

Ogni regola di registrazione è costituita da una riga divisa in due campi separati da spazi o tabulatori. Però è possibile comunque spezzare la riga usando il carattere "\" per proteggere, come sulla riga di comando, un a capo, nel qual caso gli spazi iniziali sulla riga seguente verranno ignorati, cosa che consente di formattare in maniera più leggibile regole complesse. Il primo campo è detto *selettore*, e serve a selezionare i messaggi per i quali si applica quanto indicato dal secondo, detto *azione*. Il *selettore* supporta a sua volta una sintassi abbastanza complessa, ma nella sua forma più semplice è costituito da due parti, il *servizio* (detto anche *facility*) e la *priorità*, separate da un punto.

Il *servizio* identifica una categoria di servizi di sistema per conto dei quali si vuole registrare il messaggio e viene specificato tramite una delle parole chiave riportate in tab. 3.4, dove sono elencati i servizi standard predefiniti. Oltre a quelli di tab. 3.4 ci sono una serie di servizi ausiliari, identificati dalle parole chiave da `local0` a `local7`, che sono lasciati a disposizione dell'utente per un uso non specifico.

Si tenga presente comunque che il *servizio*, così come la *priorità*, sono proprietà generiche definite dal sistema del *syslog*, che vengono specificate nelle funzioni di registrazione usate dai

[13] questi sono gestiti in maniera diversa da quelli dei programmi, per chi fosse interessato ad approfondire i dettagli si rimanda alla sez. 8.1.5 di [GaPiL].

Servizio	Significato
auth	servizio identico a authpriv, deprecato.
authpriv	messaggi relativi ad autenticazione e sicurezza.
cron	messaggi dei demoni per l'esecuzione differita (atd e cron).
daemon	demoni di sistema che non hanno una categoria di servizio a se stante.
ftp	servizio FTP (*File Transfer Protocol*, vedi sez. 7.7.2).
kern	messaggi del kernel.
lpr	messaggi dei servizi di stampa.
mail	messaggi dei demoni di gestione della posta elettronica.
mark	messaggio di marcatura a uso interno.
news	messaggi del servizio di gestione di USENET (la rete dei gruppi di discussione).
security	sinonimo di auth.
syslog	messaggi interni generati da syslogd.
user	messaggi generici a livello utente.
uucp	messaggi del sistema UUCP (*Unix to Unix CoPy*, un meccanismo di comunicazione precedente Internet).

Tabella 3.4: I servizi standard in cui sono classificati i messaggi del *syslog*.

programmi ed inserite direttamente nel contenuto dei messaggi, sono pertanto sempre presenti e completamente indipendenti dallo specifico programma che si usa per gestire i log.

I valori possibili per specificare una *priorità* sono indicati in tab. 3.5, in ordine crescente dalla più bassa alla più alta. Questi valori identificano l'importanza del messaggio, nel nostro caso tutti i messaggi di priorità superiore od uguale a quella indicata nella regola di selezione verranno registrati.

Priorità	Significato
debug	messaggio di debug.
info	messaggio informativo.
notice	situazione normale, ma significativa.
warning	avvertimento.
warn	sinonimo di warning, deprecato.
err	condizione di errore.
error	sinonimo di err, deprecato.
crit	condizione critica.
alert	si deve intervenire immediatamente.
emerg	il sistema è inusabile.
panic	sinonimo di emerg, deprecato.

Tabella 3.5: Le varie priorità dei messaggi del servizio di *syslog*.

Oltre a queste parole chiave che identificano servizi e priorità nel file di configurazione vengono riconosciute alcune estensioni; un asterisco "*" seleziona o tutti i servizi o tutte le priorità, mentre la parola chiave none li esclude tutti. Inoltre usando come separatore il carattere "," si può elencare una lista di servizi per la stessa priorità o una lista di priorità per un certo servizio. Si possono poi associare più selettori ad una stessa azione separandoli con il carattere ";". Due ulteriori estensioni di questa sintassi sono date dal carattere "=", che permette di registrare solo i messaggi aventi esattamente quella specifica priorità, e dal carattere "!", che permette di escludere i messaggi di una specifica priorità o servizio.

L'*azione* specificata come secondo campo serve ad indicare come si vuole che siano registrati i messaggi. Il caso più comune è quello in cui essi devono essere scritti su un file (quello che si chiama un *file di log*), che dovrà essere specificato tramite un pathname assoluto. Al nome si può premettere in maniera opzionale un carattere "`-`" per impedire che il contenuto del file venga sincronizzato all'arrivo di ogni messaggio, lasciando spazio per la bufferizzazione degli stessi, in tal caso il demone è in grado di accorgersi di eventuali ripetizioni di un messaggio precedente e segnalare queste ultime invece di ripetere tante volte lo stesso messaggio. Si tenga presente che i messaggi vengono sempre scritti in coda al contenuto corrente, così da non sovrascrivere messaggi precedenti.

Oltre alla scrittura su file `rsyslogd` permette di effettuare le registrazioni in diverse altre modalità. Se si premette un carattere "`|`" al nome di un file si indica che si sta facendo riferimento ad una fifo (vedi sez. 1.2.1) da cui un altro programma potrà rileggere i dati. Si può inoltre mandare l'output su una *console* specificando come file quello di un dispositivo a terminale (ad esempio `/dev/tty10`). Si possono anche mandare i messaggi a liste di utenti, identificati per username separati da virgole, e questi li riceveranno sui terminali su cui sono collegati. Infine il carattere "*" fa sì che i messaggi siano inviati a chiunque sia collegato sul sistema.

Una delle caratteristiche più interessanti del *syslog* è che si possono mandare tutti i messaggi ad una macchina remota. Questo si fa usando il carattere "`@`" seguito dal nome della macchina di destinazione; si dovrà specificare cioè l'*hostname* della destinazione o il suo indirizzo di rete, (qui si da per scontata la conoscenza di questi concetti, che tratteremo in dettaglio solo al cap. 7). Se su quella macchina è stato predisposto un `syslogd` abilitato all'ascolto via rete questo riceverà tutti i messaggi. Si può realizzare così un macchina dedicata solo a questo servizio, in modo da proteggere i file di log, che spesso possono contenere informazioni preziose utili in caso di intrusione (ovviamente detta macchina deve essere ben protetta).

Un esempio per le regole utilizzate nel file `/etc/rsyslog.conf` è il seguente estratto, che si è ripreso dal file che viene installato di default su una Debian Squeeze:

/etc/syslog.conf

```
auth,authpriv.*                 /var/log/auth.log
*.*;auth,authpriv.none          -/var/log/syslog
#cron.*                         /var/log/cron.log
daemon.*                        -/var/log/daemon.log
kern.*                          -/var/log/kern.log
lpr.*                           -/var/log/lpr.log
mail.*                          -/var/log/mail.log
user.*                          -/var/log/user.log

#
# Logging for the mail system.  Split it up so that
# it is easy to write scripts to parse these files.
#
mail.info                       -/var/log/mail.info
mail.warn                       -/var/log/mail.warn
mail.err                        /var/log/mail.err
...
#
# Some 'catch-all' logfiles.
#
*.=debug;\
        auth,authpriv.none;\
```

```
        news.none;mail.none     -/var/log/debug
*.=info;*.=notice;*.=warn;\
        auth,authpriv.none;\
        cron,daemon.none;\
        mail,news.none          -/var/log/messages
#
# Emergencies are sent to everybody logged in.
#
*.emerg                         *
...
```

In questo caso si noti come tutti i messaggi relativi ai *servizi* `auth` e `authpriv`, `daemon`, `kern`, `lpr`, `mail`, `user` e `uucp` vengano salvati su singoli file, mentre sul file `syslog` vengono mandati tutti i messaggi eccettuati quelli relativi ad `auth` e `autpriv`. Inoltre sono trattati a parte i messaggi relativi al sistema di posta, dove per ciascuna priorità viene usato un file diverso.

Vengono poi definiti alcuni file di log per i messaggi generici, come `messages` in cui si richiede l'invio dei messaggi solo delle priorità `info`, `notice` e `warn` escludendo una serie di servizi i cui messaggi sono archiviati altrove. Infine si noti come le emergenze (fra queste ci sono ad esempio i messaggi di avviso dello spegnimento del sistema) vengano stampate su tutti i terminali attivi.

Come accennato oltre alle regole di registrazione riportate nel precedente esempio, `rsyslogd` supporta delle ulteriori direttive specifiche per controllare le funzionalità aggiuntive non presenti nel vecchio `syslogd`, che si riconoscono in quanto iniziano tutte con il carattere "`$`" e prevedono una parola chiave seguita da un valore separato da spazi. Queste possono anche essere assenti, nel qual caso il valore di default consente di riottenere un comportamento compatibile con quello di `syslogd`.

In particolare si possono attivare i moduli che forniscono il supporto per queste funzionalità con la direttiva `$ModLoad` seguita dal nome del modulo da caricare, il default prevede che siano attivi i moduli `imuxsock` per la ricezione dei messaggi locali e `imklog` per la ricezione dei messaggi del kernel. Se si vuole abilitare la ricezione dei messaggi via rete (equivalente alla opzione `-r` di `syslogd`) occorre abilitare anche il modulo `imudp`.[14]

Priorità	**Significato**
`$ModLoad`	carica un modulo di estensione.
`$FileOwner`	imposta il proprietario dei file creati.
`$FileGroup`	imposta il gruppo dei file creati.
`$FileCreateMode`	imposta i permessi per i file creati.
`$DirCreateMode`	imposta i permessi per le directory create.
`$Umask`	imposta la *umask* per la creazione dei file.
`$IncludeConfig`	include ulteriori file di configurazione (supporta la sintassi del *filename globbing*).

Tabella 3.6: Le principali direttive specifiche di `rsyslogd`.

Altre direttive sono quelle che consentono le modalità con cui vengono creati i file di log, o di includere altri file di configurazione a partire da una directory (in genere `/etc/rsyslog.d/`)

[14] la ricezione di default avviene con il protocollo UDP sulla porta 514, ma questa può essere cambiata con la direttiva `$UDPServerRun`, per le spiegazioni su protocolli e porte ed i concetti relativi alla rete, che si son dati per acquisiti, si faccia riferimento a sez. 7.2.

usate in genere dai singoli pacchetti per fornire configurazioni specifiche. Un elenco delle altre principali direttive è illustrato in tab. 3.6.

Infine qualora si voglia poter utilizzare il sistema di *syslog* dalla shell (ad esempio all'interno di uno script) si può utilizzare il comando `logger`, che consente di registrare un messaggio. Il comando prende come argomento il testo del messaggio, e di default questo verrà registrato sul servizio `user` con priorità `notice`, si può però usare l'opzione `-p` per impostare una diversa priorità (o servizio) che prende un parametro nella forma *`servizio.priorità`*. Per tutti i dettagli sul comando e le altre opzioni si consulti al solito la pagina di manuale.

Benché `rsyslogd` sia il principale sostituto del vecchio `syslogd`, esistono delle alternative, una di queste, che ha ottenuto una certo successo è `syslog-ng`, che supporta anche una serie di estensioni del formato e la registrazione di alcuni dati strutturati, ma la sua incompatibilità con le configurazioni del precedente `syslogd` lo rende meno diffuso. Inoltre con il passaggio come nuovo programma di avvio a `systemd` (vedi sez. 5.3.6) alcune distribuzioni stanno iniziando ad adottare come default `journald`, il sistema di registrazione integrato in `systemd`.

La scelta in tal caso è molto controversa, in quanto `journald` salva i dati su un file binario che può essere riletto solo con l'apposito programma `journalctl`, su cui torneremo in sez. 5.3.6. Questa scelta consente di salvare molte più informazioni rispetto al *syslog*, (in particolare i dati relativi ad utenti e privilegi associati ai processi) sulle quali è possibile effettuare ricerche e selezioni, offrendo funzionalità non disponibili nel sistema tradizionale. D'altra parte espone a problemi di accesso ai dati sia perché la stabilità del formato non è garantita, per cui non è detto sia semplice accedere a vecchie registrazioni, sia perché non è garantita l'integrità delle registrazioni ed in caso di corruzione del file si possono perdere informazioni. Per questo motivo è opportuno usare `journald` come intermediario, per sfruttarne le funzionalità avanzate, ed inoltrare comunque i log (cosa che il programma supporta) ad un demone di *syslog* tradizionale.

3.2.4 Il sistema di rotazione dei file di log

I file di log sono una risorsa di grande valore in quanto i messaggi provenienti dai vari servizi che vi sono registrati possono essere di importanza fondamentale per capire le ragioni di eventuali malfunzionamenti. Il problema è che `rsyslogd`, come abbiamo appena detto, scrive sempre in modalità *append*, aggiungendo i nuovi messaggi in coda ai file, ma il problema si pone anche in quei casi, come avviene per i demoni di rete più complessi, in cui i file di log vengono scritti direttamente dal singolo programma. Questo significa che la dimensione di questi ultimi tende a crescere indefinitamente e si rischia così che la partizione su cui risiede la directory `/var/log` si riempia.

Per risolvere questo problema, e semplificare la gestione dei file di log, è stato creato un apposito programma, `logrotate`, che lanciato su base periodica gestisce la "*rotazione*" dei file di log effettuando operazioni come la loro compressione, la rimozione delle versioni troppo vecchie, e l'invio di messaggi di avviso all'amministratore per posta elettronica.

Il comando deve essere lanciato specificando come argomento un file di configurazione che specifica le modalità di rotazione. Se si usa l'opzione `-v` il comando stamperà una serie di messaggi durante le operazioni di rotazione. L'opzione `-d` permette di eseguire il comando in modalità di *debug* (ed implica `-v`), mentre l'opzione `-f` forza il comando ad eseguire una rotazione anche quando questa non è in programma. Al solito per l'elenco completo delle opzioni si può fare riferimento alla pagina di manuale.

Il formato del file di configurazione prevede la presenza di una direttiva per riga. Queste possono essere specificate direttamente nel file e vengono prese come impostazioni globali da applicare in maniera generica a tutti i file. È poi possibile indicare delle opzioni specifiche per un singolo file scrivendo al posto di una direttiva il pathname assoluto di quest'ultimo, seguito da un blocco delimitato da parentesi graffe contenente le direttive da applicargli, specificate sempre una per riga. Infine, come per gli altri file di configurazione, le linee vuote o che iniziano per "#" vengono ignorate.

Direttiva	Significato
`daily`	effettua una rotazione giornaliera.
`weekly`	effettua una rotazione settimanale, un file viene ruotato se è passata più di una settimana dall'ultima rotazione.
`monthly`	effettua una rotazione mensile, i file vengono ruotati la prima volta che il comando viene eseguito in un certo mese.
`size`	effettua la rotazione al superamento di una certa dimensione da parte del file di log (invece che su un periodo temporale); richiede che si specifichi la dimensione come argomento (supporta i suffissi `k`, `M` e `G`).
`rotate`	specifica il numero di copie (da passare come argomento) dei file di log che devono essere mantenute in una successione di rotazioni.
`include`	legge i file passati come parametri, o se trattasi di directory, tutti i file presenti all'interno di questa che non abbiano nel nome una delle estensioni vietate (indicate tramite la direttiva `tabooext`).
`create`	crea un nuovo file di log vuoto immediatamente dopo aver eseguito la rotazione del precedente assegnandogli un insieme di permessi, un proprietario ed un gruppo proprietario, che devono essere specificati come argomenti.
`tabooext`	permette di definire una lista di estensioni e caratteri vietati nei nomi dei file da includere con la direttiva `include`; questi devono essere passati come lista separata da virgole. Un carattere `+` indica di aggiungere la lista ai caratteri già esclusi, di default questi sono `.rpmorig`, `.rpmsave`, `.dpkg-dist`, `.dpkg-old`, `.dpkg-new`, `.disabled`, `,v`, `.swp`, `.rpmnew`, e `~`.
`compress`	comprime le vecchie versioni dei file di log usando `gzip`.
`missingok`	se il file di log è assente non dà errori.
`prerotate`	prima di effettuare la rotazione esegue tutti i comandi inseriti nelle linee seguenti la direttiva fino a quando non incontra una direttiva `endscript`.
`postrotate`	dopo aver effettuato la rotazione esegue tutti i comandi inseriti nelle linee seguenti la direttiva fino a quando non incontra una direttiva `endscript`.
`endscript`	termina una direttiva `postrotate` o `prerotate`.

Tabella 3.7: Le principali direttive usate nel file di configurazione di `logrotate`.

Una direttiva particolare è `include` che permette di includere altri file all'interno della configurazione. Il suo uso più interessante è però quello in cui viene usata per includere una directory. In tal caso infatti vengono inclusi tutti i file in essa contenuti a parte quelli con alcuni caratteri o con certe estensioni che li identificano come copie, definibili con la direttiva `tabooext` (vedi tab. 3.7). Questo consente ad ogni servizio che gestisce autonomamente i suoi file di log di usare un suo file di configurazione per `logrotate` con delle impostazioni personalizzate, semplicemente scrivendolo in una directory comune.

Altre due direttive particolari sono `prerotate` e `postrotate`, che consentono di eseguire una serie di comandi prima e dopo la rotazione. In genere le si usano per fermare prima, o far ripartire poi, i servizi di cui si vogliono ruotare i file di log (o per eseguire compiti analoghi). Entrambe le direttive prevedono che vengano eseguiti i comandi inseriti nelle righe ad esse seguenti, fino a

quando non si incontra la direttiva endscript.

Si ricordi infatti quanto detto in sez. 1.2.2 riguardo la cancellazione dei file: un programma attivo fa riferimento in maniera diretta ai file che ha aperto, cosa che vale anche per i file di log, con i file descriptor, per cui anche se si cancella o rinomina il file su disco il programma, usando il file descriptor, continuerà a scrivere sul file originale usando l'*inode*. Per questo motivo in genere è necessario prima fermare il programma e poi farlo ripartire, oppure dirgli esplicitamente di chiudere i file di log, cosa che alcuni programmi fanno se gli si manda un apposito segnale (in genere SIGHUP), così che al riavvio (o alla riapertura degli stessi) vada a scrivere sulla nuova versione.

La rotazione di un file può essere specificata sia per intervalli temporali (con le direttive daily, weekly e monthly) che sulla base del superamento di una certa dimensione (con la direttiva size). Quando il comando viene invocato viene controllato se la condizione è soddisfatta e solo in questo caso (a meno di non aver specificato l'opzione -f) la rotazione viene eseguita. Le altre principali direttive sono riportate in tab. 3.7, la descrizione completa di tutte le direttive e del loro significato si trova nella pagina di manuale del comando, accessibile con man logrotate.

Benché sia possibile lanciare il comando a mano, gran parte delle distribuzioni invocano logrotate fra le operazioni giornaliere eseguite da cron. In quasi tutti i casi negli script viene usato come standard il file di configurazione /etc/logrotate.conf mentre ulteriori file di configurazione di singoli servizi sono mantenuti nella directory /etc/logrotate.d/.

In genere si mettono in logrotate.conf solo alcune opzioni generali, le opzioni specifiche per ciascun servizio vengono messe nella directory /etc/logrotate.d/. In questo modo un pacchetto di installazione di un qualunque programma che ha bisogno di produrre dei log e ruotarli può inserire in questa directory un opportuno file di configurazione. Il contenuto della directory viene poi incluso tramite una direttiva include presente in logrotate.conf così che non ci sia necessità di modificare quest'ultimo ogni volta che si installa o disinstalla un pacchetto software.

Un esempio del contenuto di /etc/logrotate.conf, tratto dalla versione installata di default su una Debian Squeeze, è il seguente, i commenti spiegano in maniera molto chiara il significato delle varie opzioni:

/etc/logrotate.conf

```
# see "man logrotate" for details
# rotate log files weekly
weekly

# keep 4 weeks worth of backlogs
rotate 4

# create new (empty) log files after rotating old ones
create

# uncomment this if you want your log files compressed
#compress

# RPM packages drop log rotation information into this directory
include /etc/logrotate.d

# no packages own wtmp or btmp -- we'll rotate them here
/var/log/wtmp {
    monthly
    create 0664 root utmp
```

```
}

/var/log/btmp {
    missingok
    monthly
    create 0664 root utmp
    rotate 1
}

# system-specific logs may be configured here
```

In questo esempio si richiede di ruotare i file settimanalmente e di mantenere quattro copie degli stessi. Poi si dice di leggere tutto il contenuto della directory `/etc/logrotate.d` per ulteriori configurazioni, infine si cambiano le impostazioni di default per i due file `/var/log/wtmp` e `/var/log/btmp` che non fanno riferimento a nessun pacchetto (contengono i dati delle sessioni degli utenti) per i quali si richiede una rotazione mensile invece che settimanale.

3.3 L'*X Window System*

In questa sezione tratteremo le problematiche principali relative all'uso e alla gestione di base dell'interfaccia grafica nei sistemi Unix. In particolare approfondiremo le caratteristiche principali dell'*X Window System*, l'infrastruttura che fornisce l'interfaccia grafica, di come gestirlo e come configurarlo, con una breve panoramica finale sui programmi per l'accessibilità.

3.3.1 Introduzione a *X Window*

Benché in realtà esistano diverse alternative per la realizzazione di applicazioni in ambiente grafico, sia attraverso l'uso diretto del *framebuffer* o dell'hardware video, sia attraverso opportuni *toolkit* grafici, sia attraverso sistemi a finestre alternativi (che non hanno mai avuto una diffusione significativa), sia attraverso una reimplementazione ed estensione come *Wayland* (`http://wayland.freedesktop.org/`), a tutt'oggi le interfacce grafiche normalmente utilizzate in un sistema GNU/Linux (ed in generale in un qualunque sistema Unix) si basano tutte sull'*X Window System*, nome spesso abbreviato in *X11*, dalla versione finale del protocollo o ancora più brevemente in "*X*".

Il fatto che esistano delle alternative complete per l'uso dell'interfaccia grafica ci mette immediatamente di fronte ad una delle principali differenze che c'è fra un sistema unix-like come GNU/Linux ed altri sistemi operativi: l'interfaccia grafica non fa parte del kernel, e non ha nessuna caratteristica privilegiata, ma viene eseguita da un programma in user space come un qualunque altro programma. Tutto il sistema grafico in sostanza non è che un altro strato che viene posto sopra il kernel, che può essere tranquillamente sostituito da uno strato diverso, e che può anche essere eliminato tutte le volte che non serve (ad esempio sui server).

Il punto critico è che in realtà il programma che implementa questa interfaccia a lungo ha richiesto un accesso diretto all'hardware della scheda grafica tramite degli appositi *agganci* forniti dal kernel e questo è stato causa di instabilità con l'uso di schede grafiche per le quali i driver di supporto erano sperimentali (o proprietari). Inoltre questa architettura comportava una serie di problemi, fra cui quello di dover eseguire il cuore l'interfaccia grafica con privilegi amministrativi,

per cui nelle versioni più recenti del kernel una serie di funzionalità che prima facevano parte di *X Window* sono state portate all'interno del kernel e sono state definite delle interfacce più controllate per l'uso delle schede grafiche, che consentono anche una migliore interazione con le altre funzionalità che erano già gestite dal kernel, come la console virtuale.

L'interfaccia grafica di *X Window System* nasce a metà degli anni '80 con il progetto *Athena* al MIT. L'intento era di fornire un ambiente grafico per i sistemi UNIX, e costituisce una delle prime interfacce grafiche realizzate su un sistema operativo. La caratteristica principale di *X*, che a tutt'oggi lo distingue dalle interfacce grafiche presenti in altri sistemi operativi, è che oltre a fornire le usuali funzionalità per disegnare finestre ed altri elementi grafici su uno schermo, il sistema è completamente trasparente rispetto alla rete. Questo significa che *X Window* è in grado di operare allo stesso modo sia in locale, interfacciandosi direttamente all'hardware grafico della macchina su cui si sta operando, sia in rete, inviando attraverso di essa le informazioni per disegnare la grafica di una applicazione su una macchina remota.

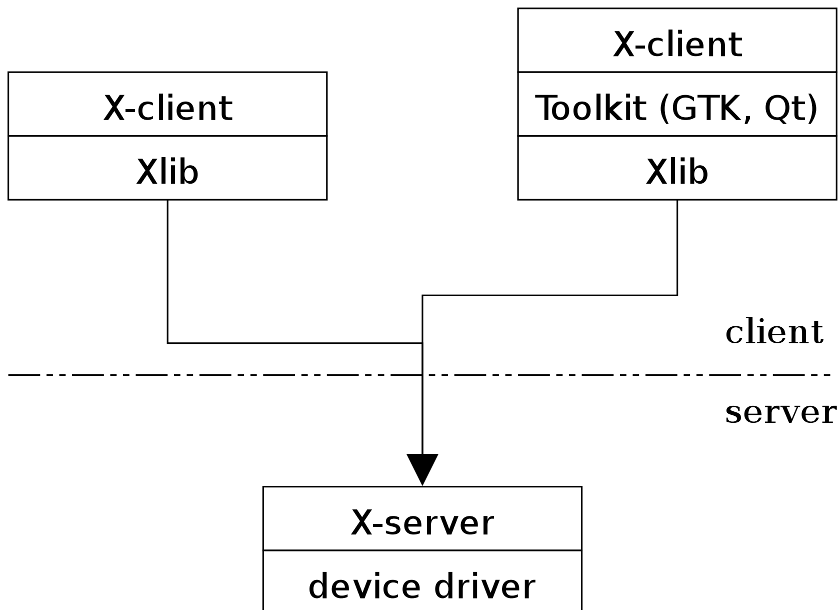


Figura 3.1: Schema di funzionamento del protocollo X11

L'architettura di *X Window* infatti definisce una relazione *client-server*, illustrata sommariamente in fig. 3.1 fra una applicazione ed il suo *display*, fornendo una serie di funzioni di libreria (tramite le *Xlib*) che separano completamente l'applicazione (detta per questo *X-client*) dal programma che si incaricherà di disegnarne le finestre (l'*X-server*). È questa separazione che consente di definire un protocollo di comunicazione (l'*X protocol*), fra client e server, che rende l'intera interfaccia trasparente rispetto alla rete.

Si tenga presente però che in questa architettura client e server hanno posizione invertita rispetto al solito. Infatti di solito è il client che viene eseguito localmente per interrogare un server remoto, che fornisce delle risposte. In questo caso sono dei client remoti (le applicazioni

X-client) che inviano le informazioni al server locale (l'*X-server*) perché questo ne disegni le finestre sulla propria macchina.

Fino a qualche anno fa la versione dell'*X Window System* più utilizzata su GNU/Linux era XFree86 nato dal porting su PC del codice originale. Questo infatti è sempre stato rilasciato con licenza libera e da sempre sono esistite diverse realizzazioni,[15] infatti la *licenza X11*, che venne creata per questo progetto e poi utilizzata in seguito anche per molti altri, è una licenza non-*copyleft* che consentiva ai vari distributori di sistemi unix-like di realizzare le proprie versioni proprietarie. Anche XFree86 aveva adottato questa licenza, ma dall'inizio del 2004, avendo effettuato un cambiamento di licenza che lo rendeva incompatibile con la GPL, la sua diffusione si è progressivamente ridotta fino alla completa scomparsa. Ma già prima di allora c'erano state parecchie critiche alla gestione del progetto, ed in particolare alla scarsa apertura nell'accettare contributi esterni, e nel restare legati ad una costruzione monolitica dell'insieme del server e delle librerie, che rendeva poco flessibile lo sviluppo.

Il cambiamento di licenza è stata l'occasione che ha fatto sì che questo malcontento desse luogo all'avvio di un progetto di sviluppo alternativo. Negli anni precedenti infatti si era avuta una significativa conversione dell'*X-Consortium* (il titolare dei diritti dell'implementazione originale di *X Window*) da consorzio di industrie a fondazione aperta anche alla partecipazione diretta di singole persone. Ne sono così entrati a far parte i principali sviluppatori che non erano d'accordo con il cambio di licenza e la gestione del progetto XFree86, e si è ripartiti dall'ultima versione di quest'ultimo rilasciata con la licenza originale, per la creazione delle nuove versioni del server X dell'*X-Consortium*, quello che oggi va sotto il nome di *X.org* (si veda `http://www.x.org/`).

Il cambio di gestione del progetto ha avuto un notevole effetto di rilancio dello sviluppo di *X Window*; in particolare nel Dicembre 2005 è stata creata, dopo più di dieci anni, una nuova *major release* del protocollo, la X11R7, che oltre ad introdurre una serie di estensioni, è stata la prima con una separazione delle varie componenti (server, librerie, driver delle schede grafiche) per consentire uno sviluppo più rapido e flessibile. Ad oggi sono già state rilasciate varie versioni del server con il nome *X.org*, che è quello adottato ormai universalmente da tutte le distribuzioni di GNU/Linux.

3.3.2 La configurazione del server X

Il primo passo per utilizzare l'interfaccia grafica su GNU/Linux è configurare il server X sulla propria macchina, perché questo possa disegnare le finestre e ricevere i dati da tastiera e mouse (oltre che dai client). Tutte le distribuzioni mantengono i vari file di configurazione dell'*X Window System* in una directory ad essi dedicata, sotto `/etc/X11`, e con *X.org* il file di configurazione del server X è `xorg.conf`.

Configurare il server X significa fornirgli un file di configurazione adeguato, che lo metta in grado di utilizzare l'hardware necessario. Fino a qualche anno fa questo veniva fatto automaticamente all'interno della procedura di installazione del sistema, in genere attraverso un opportuno programma in grado di determinare automaticamente i parametri necessari e generare il file sulla base dell'hardware della macchina.

A partire dalla versione 7.2 di *X.org* il server X è in grado di auto-configurarsi completamente e funzionare nella maggior parte dei casi anche senza avere un file di configurazione, utilizzando i

[15] un progetto interessante ad esempio è *TinyX*, una versione ottimizzata per il minor consumo possibile di risorse, da utilizzare nei sistemi embedded.

default ritenuti appropriati secondo quanto rilevato nell'hardware. Questo significa che in genere il suddetto file `xorg.conf` non è neanche più presente e deve essere creato solo in quei rari casi in cui il meccanismo di autoconfigurazione non funziona o in cui si vogliono modificare i default.

In questo caso si deve tener presente che, oltre a quelle relative all'hardware, la configurazione manuale necessita anche di altri parametri, come le informazioni relative ai font per il disegno dei caratteri o ai nomi dei colori, che però in genere sono predeterminate dall'installazione dei relativi pacchetti e vengono impostate automaticamente. Si può comunque generare uno scheletro del file di configurazione lanciando il server con `X -configure`.

Le informazioni usate nella configurazione del server X sono quelle che riguardano le varie componenti utilizzate dallo stesso, e cioè mouse, tastiera, scheda video e monitor. Si tenga presente che fin dalla sua nascita X11 ha sempre supportato la presenza di schermi multipli, e che nelle versioni recenti sono presenti pure le varie estensioni che permettono di "*spalmare*" un desktop su più schermi.

Il file di configurazione del server X è diviso in varie sezioni; ciascuna sezione inizia con la parola chiave `Section` seguita dal nome che la identifica, ed è conclusa dalla parola chiave `EndSection`. All'interno di queste sezioni dovranno poi essere specificate le opzioni ed i valori delle configurazioni (dipendenti dal tipo di sezione); la sintassi generica di una sezione sarà pertanto analoga a qualcosa come:

```
Section  "SectionName"
    SectionEntry
    ...
EndSection
```

I nomi delle principali sezioni sono riportati in tab. 3.8, l'elenco completo può essere trovato nella rispettiva pagina di manuale (la sintassi del file di configurazione è identica per XFree86 e *X.org*). Le sezioni hanno poi una loro gerarchia; alcune di esse devono essere necessariamente specificate, e quelle di livello più alto possono usare valori definiti in quelle di livello più basso.

Al livello più alto c'è la sezione `ServerLayout`, che serve a collegare insieme i dispositivi di ingresso e di uscita che vengono usati dal server in una sessione X11. Un singolo dispositivo di ingresso deve essere descritto in una sezione `InputDevice`, mentre i dispositivi di uscita devono essere collegati in una sezione `Screen`, che comprende una sezione `Device` che descrive una scheda video ed una sezione `Monitor` che descrive il monitor ad essa collegato, queste possono essere anche più di una se ci sono più monitor collegati alla stessa scheda. Le altre sezioni sono invece indipendenti.

Ciascuna sezione comprende una serie di direttive, proprie della sezione, che vanno espresse su una singola riga. Queste sono introdotte da una parola chiave seguita da uno o più valori, che hanno un formato comune per tutte le sezioni e possono essere di tre tipi diversi: numeri interi, numeri reali e stringhe. Gli interi si possono esprimere sia in forma decimale che esadecimale (apponendo il suffisso `0x`), le stringhe devono sempre essere comprese fra virgolette.

Inoltre per tutte le sezioni esiste la direttiva generica `Options` che specifica il valore di una opzione; questa richiede sempre due argomenti di tipo stringa, il primo dei quali specifica il nome dell'opzione ed il secondo il valore. La stringa che specifica quest'ultimo viene poi interpretata ed il suo contenuto viene classificato in altrettanti tipi di valore, i numeri (reali o interi, secondo il formato visto per gli argomenti generici) verranno interpretati come tali, le parole chiave `yes/no`, `on/off`, `true/false` esprimeranno invece dei valori logici, mentre aggiungendo ad un

Nome	Contenuto
`Files`	indica i pathname di file o directory contenenti informazioni usate dal server come le directory dei font o i file con le descrizioni dei colori.
`ServerFlags`	indica le opzioni generali che controllano globalmente il comportamento del server, di norma non viene utilizzato e valgono i valori di default delle stesse.
`Module`	indica i moduli aggiuntivi che devono essere caricati dinamicamente (e che forniscono estensioni come le accelerazioni 3D e il rendering dei font truetype).
`InputDevice`	contiene la descrizione di un dispositivo di input (mouse, tastiera, tavoletta grafica, ecc.); ne deve essere specificata una per singolo dispositivo.
`Device`	contiene le informazioni di configurazione di una scheda grafica, ne deve essere specificata una per ogni scheda disponibile.
`Monitor`	contiene le informazioni di descrizione di un monitor, ne deve essere specificata una per ogni monitor disponibile.
`Modes`	contiene la descrizione di una modalità video (corrisponde a specificare risoluzione, frequenza di refresh e profondità di colore).
`Screen`	contiene la definizione di uno *schermo*, combinando una scheda video ed un monitor.
`ServerLayout`	contiene la configurazione generale del server.
`DRI`	contiene le specifiche per il *Direct Rendering Interface* un'interfaccia diretta alle accelerazioni hardware delle schede grafiche.

Tabella 3.8: I nomi delle varie sezioni del file di configurazione del server X.

numero uno dei suffissi `Hz`, `kHz`, `MHz`, ecc. si potranno specificare delle frequenze; il resto verrà interpretato come stringa (l'elenco completo dei tipi di valori è comunque riportato nella pagina di manuale). Inoltre usando il prefisso `No` nello scrivere il nome di una opzione si specificherà per essa direttamente un valore logico pari a `no`, `off` o `false`.

Tratteremo qui solo le principali sezioni, indicando le principali direttive previste per ciascuna di esse; al solito per l'elenco completo si può fare riferimento alla pagina di manuale. La prima che prendiamo in esame è la sezione `Files`, che contiene la definizione di una serie di pathname che riguardano i file e le directory utilizzate dal server. È in questa sezione che si indica dove si trovano le risorse (ad esempio i font) necessari per il funzionamento del server. Le direttive possibili sono le seguenti:

`FontPath` specifica una lista di directory, separate da virgole, in cui si trovano i font usati dal server. Si può usare la direttiva più volte, e le relative directory verranno unite insieme. La direttiva richiede o un pathname assoluto ad una directory, o l'identificatore di un *font server*,[16] nella forma:

```
<trans>/<hostname>:<port-number>
```

dove `trans` indica la modalità di connessione (e vale `unix` per i socket locali e `tcp` per un collegamento remoto), `hostname` indica il nome della macchina cui si collega e `port` la porta. Un esempio tipico è:

```
FontPath        "unix/:7100"
```

[16] il font server è un servizio fornito dal programma `xfs` che permette di mantenere i font su una singola macchina, e distribuirli via rete ad altre, in modo da diminuire lo spazio disco necessario all'installazione del server o consentire l'utilizzo di font ai cosiddetti *dumb terminal* non dotati di disco; veniva utilizzato, prima del supporto diretto del rendering dei font *TrueType*, per generare automaticamente da questi ultimi dei font compatibili con il server X allora in uso.

che indica il fontserver sulla macchina locale.

RgbPath indica il pathname del file contenente il database dei colori RGB, qualora non specificato viene utilizzato il valore di default che è `/usr/lib/X11/rgb`. A partire dalla versione 7.0 infatti non viene più utilizzata la gerarchia separata `X11R6` cui si è accennato in sez. 1.2.3 e le versioni più recenti di *X.org* inoltre installano il file in `/usr/share/X11/`.

ModulePath indica il pathname delle directory in cui verranno cercati i moduli binari che forniscono le estensioni del server. Come per la direttiva `FontPath` si possono specificare più directory usando più istanze della direttiva. Qualora non sia specificato nulla viene usata di default la directory `/usr/lib/xorg/modules/`.

La gestione dei font all'interno di *X window* merita qualche dettaglio in più, l'uso della direttiva `FontPath` da sola infatti non è sufficiente, essa infatti serve ad indicare al server dove si sono installati font, ma questa installazione deve essere stata effettuata in maniera adeguata. In generale il tipo di font utilizzabile dipende dalla versione di server utilizzata ma tutte le versioni di *X.org* supportano l'uso generico di font sia *Type 1* che *TrueType*. Il server però oltre ai file che contengono i caratteri dei font ha necessità di conoscere anche le informazioni relative al nome dei font stessi, alle loro dimensioni, al tipo (normale, grassetto, corsivo, ecc.), così che questi possano poi essere utilizzati e referenziati al suo interno.

Per questo una volta installati i file dei font devono essere costruite anche tutte le informazioni necessarie al server. Un tempo questo doveva essere fatto attraverso degli opportuni programmi di utilità: una volta installati i file dei font in una directory, occorreva creare le informazioni di gestione con programmi come `mkfontdir` o `ttmkfontdir`.

Con *X.org* tutto questo non è più necessario e le informazioni relative ai font vengono gestite automaticamente tramite la libreria `fontconfig`. Questa viene controllata da una serie di file in `/etc/fonts`: la configurazione generale viene mantenuta nel file `/etc/fonts/fonts.conf`, ma eventuali personalizzazioni devono essere inserite in `/etc/fonts/local.conf` che viene mantenuto in caso di aggiornamento; i singoli utenti possono inserire le proprie in `~/.fonts.conf`. Tutti questi file sono in formato XML,[17] e se ne può trovare una descrizione dettagliata nella pagina di manuale, accessibile con `man fonts.conf`.

La sezione `ServerFlags` permette di impostare alcune opzioni generali relative al server, essa contiene solo direttive di tipo `Options` e normalmente è vuota. La sezione `Module` è usata per specificare quali moduli di estensione del server devono essere caricati. In genere prevede solo la direttiva `Load` seguita dal nome del modulo; questo è lo stesso del file corrispondente che si deve trovare all'interno del percorso indicato con `ModulePath`, tolto il `lib` iniziale e l'estensione. L'unica formulazione alternativa è usare una sintassi del tipo:

```
SubSection "extmod"
   Option  "omit XFree86-DGA"
EndSubSection
```

dove il nome del modulo è lo stesso e la direttiva `Option` viene utilizzata per passare dei parametri al modulo.

[17] *eXtendible Markup Language*, un linguaggio formale basato su marcatori.

La sezione `InputDevice` serve a specificare le caratteristiche di un singolo dispositivo di input, e normalmente se ne hanno almeno due, una per la tastiera e l'altra per il mouse, ma se ne possono specificare anche di più, in caso di presenza di più tastiere (caso non molto comune) o di più mouse (caso comune invece con i portatili, cui in genere si usa un mouse USB da affiancare al touchpad, o con l'uso di tavolette grafiche). Ogni sezione viene specificata nel formato:

```
Section "InputDevice"
    Identifier "nome"
    Driver     "dispositivo"
    Option     "..."
    ...
EndSection
```

dove devono sempre essere specificate sia la direttiva `Identifier`, che indica un nome con cui fare riferimento al dispositivo nelle altre sezioni, che la direttiva `Driver`, che ne specifica il tipo.

In particolare `Driver` permette di indicare se il dispositivo in questione viene usato per il puntamento (con il valore `mouse`) o per l'immissione di dati (con il valore `keyboard`) facendo riferimento al caso più comune di entrambe le tipologie. Le restanti direttive sono tutte da specificare nella forma di opzioni, le principali delle quali sono:

`CorePointer` di valore booleano; se viene impostata indica che il relativo dispositivo viene considerato come il mouse principale.

`CoreKeyboard` di valore booleano; se viene impostata indica che il relativo dispositivo viene considerato come la tastiera principale.

`Device` indica il file di dispositivo da utilizzare per accedere al relativo dispositivo.

`SendCoreEvents` invia gli eventi relativi ad un altro dispositivo come se fossero generati dal dispositivo principale (lo si usa sui portatili per abbinare gli eventi di un eventuale mouse USB a quelli del touchpad).

`XkbModel` indica, nel caso di una tastiera, il modello (ad esempio `pc105` per una tastiera da PC a 105 tasti).

`XkbLayout` indica, nel caso di una tastiera, la disposizione dei tasti, e corrisponde in genere al nome della relativa localizzazione (con valori come `us`, `it`, `fr`, `de`, ecc.)

`Protocol` indica nel caso di un mouse, il protocollo utilizzato (i più comuni sono `PS/2` per i mouse ordinari e `ImPS/2` per quelli dotati di rotellina).

La sezione `Device` serve a specificare le caratteristiche di una scheda grafica, e ne deve esistere almeno una (ma se ne possono avere diverse in caso di presenza di più schede grafiche). Il suo formato è identico a quello visto in precedenza per `InputDevice`, e prevede le due direttive obbligatorie `Identifier` e `Driver`, dove quest'ultima indica il nome del modulo associato al driver che gestisce la relativa scheda grafica.

Il resto della sezione prevede come nel caso precedente solo delle direttive `Option`, buona parte delle quali sono specifiche del driver della scheda grafica utilizzato e sono descritte nella

relativa pagina di manuale, cui si accede nella sezione 4 usando il nome del driver stesso. Alcune opzioni generiche, valide per tutti i driver, sono tuttavia specificate direttamente nella pagina di manuale del file di configurazione.

La sezione `Monitor` serve ad indicare le caratteristiche di un monitor, e ne deve essere presente almeno una. Il formato generale della sezione è del tipo:

```
Section "Monitor"
    Identifier "nome monitor"
    direttiva  valore
    ...
EndSection
```

dove come nei casi precedenti `Identifier` specifica una stringa identificativa del monitor (che può essere qualunque) ed è obbligatoria; le restanti direttive servono a specificare le caratteristiche del monitor, le principali sono:

`VendorName` stringa che identifica la marca del monitor.

`ModelName` stringa che identifica il modello del monitor.

`HorizSync` indica la frequenza o l'intervallo di frequenze di sincronia orizzontale supportato dal monitor (la frequenza di sincronia orizzontale è la frequenza a cui il monitor può spostare il fascio che disegna una riga orizzontale). Può essere indicato come singola lista di valori separati da virgole o come intervallo separato da un meno, e se non specificata con uno degli indicatori illustrati in precedenza è supposto essere espressa in kHz.

`VertRefresh` indica la frequenza o l'intervallo di frequenze di aggiornamento verticale supportato dal monitor (è la frequenza con cui può essere ridisegnato l'intero schermo). Come per `HorizSync` può essere indicato sia come singola lista di valori che come intervallo, espresso di default in Hz. Questo valore viene usato dal server X per determinare se una modalità video è compatibile con il monitor.

`Mode` è una direttiva speciale da scriversi su più linee (chiuse da una direttiva `EndMode`) che permette di specificare una *modalità video* (in sostanza l'insieme dei parametri che imposta la risoluzione dello schermo e la frequenza di aggiornamento dello stesso) cui viene dato il nome passato come argomento, mentre i valori da usare in termini di risoluzione e frequenze di aggiornamento vengono specificati nelle righe successive (per la sintassi si consulti la pagina di manuale).

`Modeline` è una forma alternativa della direttiva `Mode` che permette di scrivere gli stessi dati che definiscono una modalità video su di un'unica riga, per chi fosse interessato una buona guida con tutti i dettagli storici sul funzionamento ed il significato delle *Modelines* è nel *XFree86 Video Timings HOWTO*. Come per la precedente `Mode` non viene usata se non quando si voglia definire una modalità personalizzata o quando, cosa che ormai non avviene praticamente mai, il riconoscimento automatico non ha successo.

I valori per `HorizSync` e `VertRefresh` sono critici per l'uso del monitor, in quanto sbagliandoli è possibile mandare fuori sincronia lo stesso con il risultato di non vedere più nulla. I monitor recenti supportano adeguati meccanismi per notificare alle applicazioni questi ed altri dati, ed il server X provvede ad impostarli sempre in maniera automatica, per cui questi, che una volta erano i parametri più critici da impostare, ed andavano cercati fra le specifiche dei monitor o in una qualche oscura pagina del libretto di istruzione, oggi non vengono più toccati.

Gli altri parametri critici sono i valori delle modalità video espressi con `Mode` o `Modeline`, che se sbagliati di nuovo possono mandare il monitor fuori sincronia. Come accennato però queste direttive non sono più utilizzate, in quanto i valori corretti delle modalità video vengono ricavati automaticamente dalla scheda grafica che già dispone di tutti i modi standard VESA, che vengono utilizzati in base alle informazioni ottenute dal monitor o dalla risoluzione impostata dalla direttiva `Modes` nella sezione `Screen`.

Una volta definite le schede video ed i monitor presenti, questi vanno collegati fra loro oltre che con il cavo tramite una apposita sezione `Screen` che identifica l'insieme dei due, e definisce le caratteristiche dello "*schermo*" che verrà utilizzato dal server X. Il formato generale di questa sezione è il seguente:

```
Section "Screen"
    Identifier "name"
    Device     "device id"
    Monitor    "monitor id"
    direttive
    ...
    SubSection "Display"
       voci
       ...
    EndSubSection
    ...
EndSection
```

di nuovo la direttiva `Identifier` serve a fornire una stringa che identifica lo schermo in questione, ed è obbligatoria come le due `Device` e `Monitor` che servono a specificare la scheda video ed il monitor collegati allo schermo, che prendono come argomento l'identificatore utilizzato in una delle sezioni omonime. Viene spesso utilizzata anche la direttiva `DefaultDepth` che indica la profondità di colore (in numero di bit) da utilizzare di default per lo schermo quando si hanno più sottosezioni di tipo `Display`.

È inoltre obbligatoria la presenza di almeno una sottosezione `Display` che specifichi la profondità di colore (tramite la direttiva `Depth`, obbligatoriamente presente) e l'indicazione di una (o più) modalità video con la direttiva `Modes`, che prende un elenco di nomi (separati da spazi) fra quelli predefiniti nello standard VESA o indicati da una precedente direttiva `Mode` o `Modeline` presente nella corrispondente sezione `Monitor`. Uno schermo infatti può essere utilizzato a diverse profondità di colore, diverse risoluzioni e diverse frequenze di aggiornamento a seconda delle caratteristiche della scheda video e del monitor, e qualora si specifichino più modi, è pure possibile passare al volo dall'uno all'altro una volta avviato il server con la combinazione di tasti `C-M-+` e `C-M--`.

Infine la sezione `ServerLayout` serve ad ottenere una configurazione completa collegando uno o più schermi (definiti in altrettante sezioni `Screen`) con uno o più dispositivi di input (definiti in altrettante sezioni `InputDevice`). Di nuovo ne deve essere presente almeno una, nella forma generica:

```
Section "ServerLayout"
    Identifier   "nome"
    Screen       "screen id"
    ...
    InputDevice  "input device id"
    ...
    opzioni
    ...
EndSection
```

di nuovo `Identifier` indica una stringa di identificazione, mentre `Screen` e `InputDevice` servono a referenziare quali schermi e dispositivi di input inserire nel layout, usando l'identificatore specificato nelle omonime sezioni.

3.3.3 L'avvio di una sessione di lavoro sotto X

Le modalità per attivare una sessione di lavoro usando l'interfaccia grafica fornite da *X Window* sono sostanzialmente due, quella che prevede l'avvio del server X e di tutto l'ambiente grafico da *console*, che oggi però non viene praticamente mai usata, e quella che prevede il login direttamente dall'ambiente grafico, che in questo caso viene attivato direttamente nella procedura di avvio (tramite gli opportuni script) insieme al programma di gestione del login.

Il programma per avviare una sessione grafica a partire dalla *console* è `xinit`, che si usa soltanto quando non si può o non si vuole lanciare il server X nella fase di avvio del sistema, o in ambienti in cui si abbia la necessità di utilizzare diverse versioni del server X, ad esempio se, avendo più schede e più monitor, si volesse usare XFree86 per una scheda e X.org per l'altra. Dato che, come illustrato in sez. 3.3.1, per l'uso dell'interfaccia grafica occorre disporre sia del server X che degli opportuni programmi client che la realizzano, `xinit` ha la particolare caratteristica di suddividere gli argomenti che gli vengono passati in due sezioni, separate dalla sequenza "`--`".

La prima sezione è quella che indica il programma da eseguire come client del server X ed i rispettivi argomenti. Essa deve iniziare con il nome del programma, che deve essere specificato con un pathname esplicito in forma assoluta o relativa, deve cioè essere nella forma "`/path/to/prog`" o "`./prog`", altrimenti l'intera sezione sarà considerata come costituita soltanto dagli argomenti da passare al programma usato di default. In genere si usa come programma un opportuno script di avvio sessione che lancia tutti i programmi necessari a realizzare la stessa.

La seconda sezione indica invece quale deve essere il programma da usare come server X, e deve essere specificata in maniera analoga alla prima, vale a dire che se non si usa al suo inizio un pathname esplicito la sezione viene considerata come costituita esclusivamente dagli argomenti da passare al server. In genere non la si usa quasi mai (per cui la sequenza "`--`" viene omessa e si passa a `xinit` solo la prima sezione) se non per specificare qualche argomento alternativo al server X. In generale infatti non ci sono motivi per cambiare il default usato da `xinit` che è

l'esecuzione del server X con il comando "`X :0`", dove `:0` indica l'uso del primo *display* disponibile (torneremo sul concetto di *display* in sez. 3.3.4).

Una volta invocato, `xinit` prima mette in esecuzione il server X e poi esegue il programma client specificato nella prima sezione, che a questo punto potrà lanciare la sessione grafica. Quando detto programma termina, `xinit` si incarica di chiudere la sessione grafica terminando anche il server X. Si tenga presente che questo avviene anche se nel frattempo ci sono altri programmi che usano X che sono ancora attivi, lanciati per esempio dal client specificato nella prima sezione.

Se non si specifica nulla sulla riga di comando all'avvio della sessione `xinit` cercherà nella home dell'utente il file `.xinitrc` e lo eseguirà come se fosse uno script di shell, terminando la sessione (e fermando il server X) solo alla conclusione dell'esecuzione del contenuto di `.xinitrc`. Non è necessario che il file sia eseguibile (né che inizi con `#!/bin/sh`) se esiste viene lanciato come programma client, nella forma `sh ~/.xinitrc`.

Questo consente di utilizzare `.xinitrc` per inizializzare il proprio ambiente grafico, chiamando al suo interno tutti i programmi che si intendono lanciare. In assenza di uno `.xinitrc` creato dell'utente, `xinit` eseguirà il programma `xterm`, un semplice emulatore di terminale che si limita a riprodurre dentro una finestra un'interfaccia a terminale del tutto analoga a quella che si trova sulla *console*.

Se si utilizza questa funzionalità occorre però considerare che i programmi eseguiti tramite `.xinitrc` vengono avviati in sequenza: se si vuole che vengano eseguiti in maniera concorrente occorrerà lanciarli tutti in background, altrimenti fino alla terminazione di uno di essi non partirà il successivo. Questo per tutti tranne l'ultimo, altrimenti l'esecuzione di `.xinitrc` proseguirebbe con la conclusione dello script, e la conseguente terminazione della sessione. Per questo motivo in genere si lancia sempre per ultimo un *window manager* o un *session manager* (vedi sez. 3.3.4).

Normalmente `xinit` non viene mai eseguito direttamente, e per avviare una sessione grafica da *console* si usa invece `startx`. Quest'ultimo non è altro che uno script di shell che esegue `xinit` fornendogli dei default opportuni, integrandosi con le impostazioni scelte dalla propria distribuzione in modo da utilizzare gli eventuali *window manager* o *session manager* che sono stati predisposti come default.

Il comando si esegue in genere senza argomenti, se presente utilizza anch'esso il file `.xinitrc`, ma se questo è assente viene eseguita una inizializzazione di default, identica per tutto il sistema, sulla base del contenuto del file `/etc/X11/xinit/xinitrc`. Il risultato dipende da quello che è stato impostato dalla vostra distribuzione come sessione grafica di default, ad esempio nel caso di Debian detto file fa a sua volta riferimento a quanto indicato in `/etc/X11/Xsession`. Oltre al comportamento standard si possono passare degli argomenti sulla riga di comando anche a `startx`, con lo stesso formato illustrato in precedenza per `xinit`, nel qual caso si avrà anche lo stesso comportamento di quest'ultimo; ad esempio per lanciare dalla *console* una sessione in cui si usi soltanto un terminale si potrà usare:

```
startx /usr/bin/xterm -title "Terminale di prova" -geometry 80x25
```

Infine, essendo il server X un programma come gli altri, è sempre possibile lanciarne più istanze in contemporanea. In tal caso è possibile distinguere a quale istanza fare riferimento in base al cosiddetto numero di *display* (tratteremo i dettagli di questa caratteristica all'inizio di sez. 3.3.4). All'avvio del server di norma viene utilizzato il primo numero di *display* disponibile, cioè zero. Sia `xinit` che `startx` consentono di lanciare ulteriori istanze del server X, ma in

tal caso occorre specificare un numero di display diverso, che andrà passato come argomento aggiuntivo. In tal caso si tratta di specificare solo questo come argomento aggiuntivo dopo la sequenza "--", ed il valore di *display* indicato andrà a soprassedere quello di default. Così se si vuole lanciare una seconda istanza del server X che utilizzi il display 1 si dovrà eseguire il comando:

```
startx -- :1
```

La seconda modalità di avvio del server X è quella eseguita attraverso l'impiego di un *login manager* grafico. In genere perché questo avvenga dovrà essere stata opportunamente configurata la procedura di avvio (vedi sez. 5.3.5) per lanciare automaticamente detto programma. Questo ormai viene fatto nelle installazioni di default di praticamente tutte le distribuzioni, tranne il caso in cui si sia fatta la scelta esplicita di non installare l'interfaccia grafica.

Un *login manager* è semplicemente un programma che lancia il server X e poi si incarica di mostrare all'utente il cosiddetto *greeter*, una finestra di login (grafica) su cui questo può autenticarsi per entrare nel sistema ed eseguire una *sessione* di lavoro. In realtà il compito di un *login manager* è un po' più complesso, infatti è possibile sfruttare la trasparenza sulla rete del protocollo X11 per eseguire la stessa procedura di autenticazione e di avvio di una sessione di lavoro in modalità remota, cosa che richiede il supporto di un apposito protocollo, l'XDMCP (*X Display Manager Control Protocol*).

Questa funzionalità consente di presentare anche su macchine remote, che a questo punto assumeranno il compito di fare da semplici *terminali* per la sessione, la stessa finestra di login. In questo modo è possibile centralizzare le applicazioni grafiche su un unico server su cui esse saranno eseguite, e riutilizzare macchine dotate anche di scarse risorse hardware, che non sarebbero in grado di sopportare il carico di tutta l'interfaccia grafica, solo per il disegno delle finestre attraverso l'installazione del solo server X.

Nella distribuzione classica di *X Window*, quella distribuita con l'implementazione di riferimento dell'X-server, il programma che implementa il *login manager* è `xdm`. Il programma ha una interfaccia molto spartana, ma è ampiamente configurabile in quanto il suo comportamento viene governato da una serie di script di shell e dall'impostazione delle opportune "*risorse*" (le "*risorse*" di *X Window* sono trattate in sez. 3.3.4); questo ne complica la configurazione in quanto i vari aspetti del suo funzionamento sono controllati tramite molteplici file.

Benché `xdm`, per il suo legame diretto con il sistema di *X Window*, sia disponibile in maniera praticamente universale, resta alquanto primitivo per cui la gran parte delle distribuzioni GNU/Linux recenti hanno adottato dei *login manager* alternativi. Due dei più diffusi sono GDM (lo *Gnome Display Manager*) che fa parte del desktop Gnome e KDM (il *KDE Display Manager*) che fa parte del desktop KDE. Entrambi offrono una grafica molto più accattivante di `xdm` ed un sistema di configurazione molto più avanzato (e semplice da usare).

Infatti benché entrambi non necessitino della presenza dei rispettivi ambienti desktop e possano essere usati indipendentemente, vengono solitamente installati in corrispondenza a questi. Il grande vantaggio di entrambi è che, pur essendo controllati da un file di configurazione testuale, è possibile controllarli e modificarne alcune impostazioni con un apposito programma ad interfaccia grafica, che permette di attivare e controllare le loro funzionalità senza dover modificare a mano i file di configurazione.

Nel caso di GDM a partire dalla versione 2.22 il programma è stato riscritto e suddiviso in due parti, il demone `gdm` vero e proprio che lancia il server X ed esegue le operazioni di accesso

e di creazione della sessione di lavoro ed il *greeter*, che è l'applicazione grafica che visualizza la finestra di accesso (ed i relativi accessori) ed il cui compito è sostanzialmente quello di autenticare l'utente prima dell'accesso e che viene configurata a parte con il sistema di *gconf* di Gnome.[18]

Il file di configurazione principale di GDM è `/etc/gdm/gdm.conf`; il file ha il formato in stile dei *keyfile*, che abbiamo brevemente illustrato in sez. 3.1.1. Il file prevede una serie di sezioni, le principali delle quali sono elencate in tab. 3.9.

Nome	**Significato**
`[daemon]`	contiene le direttive di controllo generali del demone `gdm`, come i nomi di file directory e comandi utilizzati, utente e gruppo per conto del quale eseguire i programmi come il *greeter*, ecc. In genere è vuota mantenendo tutti i valori ai rispettivi default.
`[security]`	contiene le opzioni per la gestione della sicurezza, a volte vi si imposta `AllowRoot=true` per consentire all'amministratore di eseguire il login grafico, ma si tratta di una scelta da evitare nella maniera più assoluta, visto che sono disponibili numerose alternative per lanciare programmi ad interfaccia grafica con privilegi di amministratore senza dover far girare un intero ambiente desktop, non pensato a questo scopo, per conto dello stesso.
`[xdmcp]`	contiene la direttive per gestire il login grafico da remoto con il protocollo *XDMCP*, se lo si vuole abilitare occorre impostare la direttiva `enable=true`, le altre direttive non vengono in genere usate essendo ragionevoli i valori di default.

Tabella 3.9: Nomi delle principali sezioni di `gdm.conf`.

Insieme a `gdm.conf` il programma utilizza anche una serie di altri file e directory (riportati in tab. 3.10) posti sempre in `/etc/gdm`, che consentono una personalizzazione molto dettagliata di tutta la procedura di accesso, è comunque possibile avviare il demone richiedendo con le opportune opzioni a riga di comando l'uso un altro file di configurazione o di una directory diversa da `/etc/gdm`. Le directory elencate nella prima parte di tab. 3.10 consentono di far eseguire a `gdm` uno specifico script per ciascuna delle varie fasi di gestione della sessione, secondo quanto illustrato nella tabella stessa. Si tenga presente che gli script in questione, anche quelli relativi alla sessione, vengono eseguiti con privilegi amministrativi e non per conto dell'utente, il cui nome viene passato nella variabile di ambiente `USER`.

Tutti gli script presenti in queste directory seguono uno stesso schema di nomi, che consente di eseguire script diversi a seconda del *display* su cui è stato lanciato il server X. Infatti se `gdm` trova uno script corrispondente al numero di *display* questo verrà eseguito nel relativo punto della sequenza, altrimenti verrà eseguito, se presente, lo script `Default`. Un caso particolare resta il contenuto della directory `Init`, che viene utilizzato anche per effettuare login remoti con XDMCP, attivati dalla presenza dello script `Init/XDMCP`.

Come avvenuto per Gnome anche l'ambiente grafico KDE ha sviluppato un suo *Display Manager*, `kdm`, che fornisce funzionalità simili. In questo caso la configurazione viene mantenuta nella sottodirectory `kdm` delle configurazioni di KDE, vale a dire, a seconda della versione di KDE usata, `/etc/kde3/kdm` o `/etc/kde4/kdm`. Il file di configurazione principale è `kdmrc`, che di nuovo usa il formato dei *keyfile* descritto in sez. 3.1.1.

[18] vale a dire con una specie di riedizione del "registro" di Windows, che per quanto sia stata realizzata in maniera più sana, facendo ricorso a file di testo in XML, rende comunque quasi obbligatorio il ricorso ad un apposito programma (ad esempio `gconf-editor`) per modificare le chiavi di configurazione onde evitare di perdersi in una marea di tag.

Nome	Significato
Init/	contiene gli script da eseguire immediatamente dopo l'avvio del server X e prima del lancio del *greeter*, in genere vengono usati per lanciare programmi che si vogliono eseguire durante la schermata di accesso o per effettuare eventuali inizializzazioni prima della stessa.
PostLogin/	contiene gli script da eseguire immediatamente dopo l'autenticazione dell'utente, prima che venga eseguita qualunque operazione di inizializzazione della sessione, usati ad esempio per impostare proprietà della stessa, come la home dell'utente.
PreSession/	contiene gli script da eseguire immediatamente prima dell'avvio della sessione finale, in genere per mantenere informazioni di contabilità .
PostSession/	contiene gli script da eseguire alla chiusura di una sessione di lavoro.
modules/	contiene le configurazioni usate dai moduli che forniscono il supporto per le funzionalità di accessibilità (vedi sez. 3.3.5).
locale.alias	file contenente la lista delle localizzazioni disponibili (vedi sez. 3.1.4).
Xsession	indica lo script utilizzato per avviare la sessione, che viene eseguito dopo gli script della directory `PreSession/` per conto dell'utente.
XKeepsCrashing	indica lo script eseguito quando l'avvio del server X fallisce, viene in genere usato per invocare il programma di configurazione del server X.

Tabella 3.10: I file in `/etc/gdm` che controllano `gdm`.

Oltre alle direttive delle tre sezioni generali, riportate in tab. 3.11, `kdmrc` può contenere configurazioni specifiche da eseguire per ciascuna istanza del server X che viene lanciata (associate al relativo *display*) da mettere in altrettante sezioni dedicate nella forma `[X-*-Core]` e `[X-*-Greeter]`, dove al posto dell'asterisco deve essere indicato il nome del *display* (come indicato in sez. 3.3.4) e della classe di risorse da configurare. Per una trattazione completa della sintassi di questo file si rimanda alla lettura della documentazione ufficiale consultabile su `http://docs.kde.org/stable/en/kde-workspace/kdm/index.html`.

Nome	Significato
[General]	contiene le direttive di controllo generali del demone `kdm`.
[Xdmcp]	contiene le direttive di controllo per fornire il login grafico da remoto con il protocollo *XDMCP*.
[Shutdown]	contiene le direttive relative ai programmi da usare per gestire il riavvio ed il fermo macchina da `kdm`.

Tabella 3.11: Nomi delle principali sezioni di `kdmrc`.

Oltre al file `kdmrc` nella directory di configurazione vengono mantenuti una serie di altri file: ad esempio per lanciare il demone viene usato dallo script di avvio il file `kdm.option`, con lo stesso significato e formato dell'analogo di XDM. Inoltre nella stessa directory vengono mantenuti gli script `Xsetup`, `Xstartup`, `Xsession`, ecc. anch'essi con lo stesso significato degli analoghi di XDM. In particolare `Xsetup` viene lanciato dopo l'avvio del server X prima di far partire il *greeter* per operare sulla finestra di accesso, `Xstartup` quando l'utente si è autenticato prima lanciare la sessione per predisporre la sessione, mentre `Xsession` viene usato per avviare la sessione.

Ai due precedenti si affianca, adottato da Ubuntu che è la distribuzione desktop più diffusa, `lightdm`, un altro *display manager* nato per essere completamente svincolato rispetto all'ambiente grafico sottostante, la cui interfaccia viene disegnata a partire da del semplice codice HTML. Il vantaggio è il non dipendere da nessun toolkit grafico specifico ed avere quindi una base di

codice nettamente più ridotta, pur fornendo le stesse funzionalità di GDM, mentre il *greeter* può essere realizzato con qualunque ambiente grafico, rendendo possibile installare solo le librerie grafiche di quell'ambiente.

Il programma viene controllato dal file di configurazione `/etc/lightdm/lightdm.conf`, che di nuovo segue il formato dei *keyfile* descritto in sez. 3.1.1. A questo si aggiungono, se presenti, eventuali file `/etc/lightdm/lightdm.conf.d/*.conf` in modo da consentire ad altri pacchetti software l'installazione di configurazioni specifiche senza toccare la configurazione principale; si tenga comunque presente che `lightdm.conf` ha la precedenza su questi ultimi.

Il file prevede 4 sezioni principali, riportate in tab. 3.12, in genere le ultime due sono, per una installazione normale di una stazione di lavoro personale, vuote, ma consentono di attivare le sessioni grafiche remote o tramite il protocollo XDMCP citato precedentemente, o tramite il protocollo VNC.

Nome	**Significato**
`[LightDM]`	configurazioni generali di `lightdm`.
`[SeatDefaults]`	configurazioni relative alla sessione di lavoro per la *postazione* generica.
`[XDMCPServer]`	configurazioni per fornire una sessione di lavoro remota con il protocollo XDMCP.
`[VNCServer]`	configurazioni per fornire una sessione di lavoro remota con il protocollo VNC.
`[Seat:N]`	configurazioni specifiche alla sessione di lavoro lanciata sull'`N`-sima *postazione*.

Tabella 3.12: Nomi delle principali sezioni di `lightdm.conf`.

La prima sezione, `LightDM`, contiene le direttive di configurazione generali del programma: ad esempio con `greeter-user` si imposta l'utente per conto del quale verrà eseguito il *greeter* (il default è `lightdm`). In genere non viene utilizzata non essendovi necessità di cambiare i valori di default. La sezione più importante è `SeatDefaults` che imposta le caratteristiche di una *postazione di lavoro* (il cosiddetto *seat*) generica, mentre si possono indicare impostazioni specifiche per le singole postazioni (corrispondenti ad altrettanti *display* di X) con altrettante sezioni nella forma `Seat:N`.

Nome	**Significato**
`greeter-session`	indica il programma da lanciare come *greeter* (ad esempio `unity-greeter`, o `lightdm-gtk-greeter`).
`user-session`	indica quale sessione grafica far partire fra quelle eventualmente disponibili.
`allow-guest`	consente di creare una sessione di lavoro per un ospite, senza autenticazione.
`greeter-show-manual-login`	indica se il *greeter* deve offrire un manuale di ausilio per le opzioni di login.
`greeter-hide-users`	indica al *greeter* di non mostrare l'elenco degli utenti.
`display-setup-script`	script eseguito all'avvio del server X.
`greeter-setup-script`	script eseguito all'avvio del *greeter*.
`session-setup-script`	script eseguito al lancio della sessione.
`display-cleanup-script`	script eseguito alla chiusura del server X.
`greeter-cleanup-script`	script eseguito all'uscita dal *greeter*.
`session-cleanup-script`	script eseguito all'uscita dalla sessione.

Tabella 3.13: Principali direttive della sezione `[SeatDefaults]` di `lightdm.conf`.

É nella sezione `SeatDefaults` che si imposta quale programma lanciare come *greeter* (con la direttiva `greeter-session`), cosa usare come sessione grafica di default per gli utenti (con la direttiva `user-session`), e gli eventuali script da lanciare nelle varie fasi di accesso al sistema (con le direttive `*-setup-script`) o di uscita dallo stesso (con le direttive `*-cleanup-script`). Si sono riportate in tab. 3.13 le principali direttive disponibili per questa sezione.

3.3.4 L'uso di X Window dal lato client

Come accennato in sez. 3.3.1 qualunque programma che usi l'interfaccia grafica tramite le *Xlib* lo fa agendo come client per un server X che poi si incaricherà di disegnare sullo schermo da lui gestito gli elementi grafici indicati dall'applicazione. Per il client si pone allora il problema di come sapere a quale istanza del server X deve fare riferimento.

Per risolvere questo problema ogni server X è identificato tramite quello che viene chiamato il suo "*display name*". Considerando che il protocollo X11 è nato per essere usato anche via rete, il server X a cui il client deve rivolgersi potrebbe anche essere remoto; per questo motivo il *display name* viene espresso nella sua forma più generica con un identificativo del tipo:

```
nomehost:Nd.Ns
```

dove `nomehost` è il nome della eventuale macchina remota, che può essere espresso sia con un indirizzo IP che con un nome a dominio,[19] mentre "*Nd*" è un numero intero, detto *display number*, che corrisponde all'istanza del server lanciata (quello da passare nell'esempio di avvio del server X, che abbiamo illustrato in sez. 3.3.3). Dato che il server X supporta anche la possibilità di utilizzare in contemporanea più schermi (facendo riferimento alla stessa tastiera e allo stesso mouse), si può anche scegliere uno schermo specifico con il numero "*Ns*", detto *screen number*.

Sia *Nd* che *Ns* sono numeri che, salvo indicazioni contrarie, vengono allocati dinamicamente per grandezza crescente a partire da zero. Se non si specifica il numero di schermo si suppone che ce ne sia uno solo. Se invece non si specifica il nome della macchina si intende dire che il server X è attivo in locale, e verrà utilizzata la modalità più efficiente per comunicare con lui che passa attraverso l'uso di un socket locale (in genere è `/tmp/.X11-unix/XN`, dove "*N*" corrisponde al numero di *display*).

Il valore del *display name* viene mantenuto nella forma appena illustrata (senza l'indicazione dell'*hostname*, se si lavora in locale) in una apposita variabile di ambiente, `DISPLAY`, che è quella che viene usata dalle *Xlib* per identificare il server X con cui l'applicazione deve parlare. Molte applicazioni accettano una opzione `-display` sulla riga di comando con cui si può specificare un *display name* alternativo, altrimenti viene usato il valore della variabile di ambiente `DISPLAY`. Quando si avvia una sessione grafica locale è cura della procedura di inizializzazione della sessione definire in maniera automatica questa variabile, in modo da renderla disponibile per tutti i programmi che si utilizzeranno al suo interno; se non viene impostata qualunque client fallirà con errore del tipo: "*Can't open display*".

Finora abbiamo parlato della capacità del protocollo di poter essere utilizzato via rete, ma nella configurazione di default di un server X questa capacità viene usualmente disabilitata. Il protocollo X11 infatti di base supporta un meccanismo di sicurezza piuttosto debole, con una autenticazione del client basata sull'uso di un cookie di sessione, che ciascun client deve

[19] tratteremo in dettaglio questi argomenti, qui dati per scontati, in sez. 7.

presentare al server per poter avere accesso. I client lo ottengono utilizzando il contenuto del file indicato dalla variabile di ambiente `XAUTHORITY`. Nel caso di una sessione locale il cookie di sessione viene mantenuto di solito nel file `.Xauthority` nella home dell'utente che ha messo in esecuzione il server X. Il cookie, questo file, e la variabile `XAUTHORITY`, vengono creati al momento dell'avvio della sessione grafica.

Il problema è che una volta che si è ottenuto l'accesso al server non esistono meccanismi di limitazione o controllo, e si può compiere su di esso qualunque operazione; questo significa ad esempio che si può creare una finestra trasparente che copre tutto lo schermo e farle intercettare tutti i dati in ingresso (comprese eventuali password immesse su una emulazione di terminale). Fintanto che l'accesso è locale questo non costituisce un problema, in quanto il file `.Xauthority` non è accessibile dagli altri utenti, ma se si abilita la comunicazione via rete questo, essendo trasmesso in chiaro, può essere facilmente intercettato e riutilizzato da terzi per garantirsi un accesso completo al server X. Anche se in teoria c'è la possibilità di utilizzare meccanismi più complessi, come cookie cifrati con chiavi DES condivise o appoggiarsi a *Kerberos* (per i dettagli si consulti la pagina di manuale accessibile con `man Xsecurity`), ma questo complica notevolmente la configurazione e perciò normalmente non viene usato.

Per questo motivo il default adottato quasi universalmente è quello di utilizzare il server X soltanto in locale, a meno di non operare su reti chiuse e molto controllate (ad esempio quelle di un aula). Questo significa che su molte distribuzioni il server X viene lanciato con l'opzione `-nolisten tcp` (su Debian questa l'impostazione è usata in `/etc/X11/xinit/xserverrc`), opzione che deve essere rimossa se si vuole poter utilizzare il server via rete. Una volta messo in esecuzione il server in ascolto sulla rete, occorre poi consentire l'accesso al server X da parte dei client.[20]

Un client infatti per poter parlare con il server deve presentare il cookie di sessione, ma questo è presente solo sulla macchina su cui si è avviato il server, è pertanto necessario fornire questa informazione ai client sulla macchina remota da cui si collega, dato che i client usano di default il file indicato da `XAUTHORITY`, che essendo su un macchina diversa non potrà indicare un cookie di sessione valido.

Per modificare le modalità con cui il server autorizza i client all'accesso esiste un apposito programma, `xauth`, che permette, eseguendolo sul server, di estrarre il cookie della sessione locale, e di reimportarlo, eseguendolo sul client, in quella remota (per i dettagli sul comando ed un esempio di come usarlo si consulti la rispettiva pagina di manuale). Dato che anche con questo procedimento si ha comunque il trasferimento dei dati via rete in chiaro, pertanto senza ottenere un livello di sicurezza significativo, in genere si preferisce utilizzare un meccanismo di accesso ancora più elementare, ma più semplice da usare, che si basa solo sul consentire l'accesso al server solo in base alla provenienza delle connessioni da una determinata macchina.

Anche in questo caso l'accesso deve essere abilitato in maniera esplicita; per questo viene utilizzato l'apposito comando `xhost`, che indica al server X quali sono le macchine da cui accettare connessioni via rete senza richiedere nessun cookie di sessione. Il comando prende come argomento il nome (o l'indirizzo di rete) della macchina a cui si vuole consentire l'accesso, preceduto opzionalmente dal carattere "`+`" per indicare che esso sarà aggiunto alla lista delle macchine consentite, mentre se vi si appone il carattere "`-`" il nome verrà tolto dalla lista. Si può consentire l'accesso o rimuoverlo in maniera generica utilizzando i suddetti caratteri senza

[20]il protocollo X11 utilizza per la comunicazione via rete le porte TCP (porte e protocolli di rete sono trattati in sez. 7.2) a partire dalla 6000, ne viene utilizzata una per ciascun display attivo, il cui valore si ottiene sommando a 6000 il numero di display.

farli seguire da nessun nome, in sostanza con `xhost +` chiunque potrà inviare finestre sulla nostra macchina. Se infine si esegue il comando senza specificare nulla verrà stampata la lista delle macchine autorizzate.

Dato che alla fine, anche a causa della notevole quantità di banda richiesta, l'uso del protocollo X11 in remoto avviene quasi esclusivamente su reti locali, l'operazione che tradizionalmente si effettua per ottenere l'invio delle finestre da una macchina remota alla propria è quella in cui prima si definisce sulla macchina remota su cui gira il client un opportuno valore per la variabile `DISPLAY` che indichi la propria macchina come destinazione (ad esempio con qualcosa del tipo `DISPLAY=serverhost:0`) mentre su quest'ultima si dovrà consentire l'accesso da parte dei client sulla macchina remota, eseguendo un comando del tipo `xhost +clienthost`. Vedremo in sez. 8.3.4 come sia possibile usare questa funzionalità in maniera sicura con `ssh`, senza la necessità di nessuna di queste operazioni.

Un'altra funzionalità di *X Window* è quella di fornire un meccanismo generico per impostare le preferenze relative all'ambiente grafico di ciascuna applicazione, tramite quelle che vengono chiamate *risorse*. Il vantaggio di questa funzionalità sta nel fatto che i valori di queste *risorse* vengono mantenuti sul server, in un database interno, evitando così di doverle specificare separatamente per ciascun client. È possibile inoltre, quando si vogliono personalizzare le preferenze relative alle singole applicazioni grafiche, caricare direttamente sul server valori diversi, senza dover con questo andare a modificare le impostazioni sui client.

Il valore di una risorsa è sempre specificato tramite una stringa; le risorse poi possono a loro volta contenere altre risorse, il che comporta una organizzazione strutturata su una gerarchia ad albero in cui ogni nodo è identificato dal nome della risorsa, i singoli nodi sono separati dal carattere "`.`" e che ha come radice il nome del programma a cui la risorsa fa riferimento spesso scritto con la prima lettera maiuscola.[21] Esiste inoltre tutta una serie di risorse generiche, come quelle relative ai colori di sfondo, i caratteri, ecc. che possono venire applicate ai singoli elementi grafici di un programma, consentendo così una configurabilità estremamente dettagliata.

In genere per le risorse di ciascuna applicazione vengono mantenute delle impostazioni generali in un file a questo dedicato, che in fase di installazione viene posto in una directory comune (nelle distribuzioni recenti `/etc/X11/app-defaults/`) con il nome stesso della risorsa. I singoli utenti però possono cambiare a piacere questi valori tramite l'uso del file `.Xresources` nella propria home directory, nel quale specificare i valori che all'avvio del server andranno a soprassedere quelli di default. Se poi si vogliono cambiare gli stessi valori una volta avviato il server si può usare il comando `xrdb` che prende come argomento il file in cui sono stati scritti i nuovi valori che saranno caricati immediatamente. Un esempio di un file di configurazione delle risorse può essere il seguente:

.Xresources
```
...
emacs*Background: DarkSlateGray
emacs*Foreground: Wheat
emacs*pointerColor: LightSteelBlue
```

[21] le *risorse* si chiamano così in quanto corrispondono effettivamente a delle strutture dati che a loro volta possono contenere altre strutture dati, i campi di una di queste strutture identificano cioè un possibile valore o una ulteriore risorsa; ciascun programma definisce quali strutture vuole utilizzare per il controllo del suo aspetto grafico (il colore delle finestre, dello sfondo, il titolo della finestra, ecc.); il valore viene associato ad un campo della risorsa stessa con un certo nome, per cui viene naturale accedere all'insieme di strutture e sottostrutture attraverso una gerarchia ad albero.

```
emacs*cursorColor: LightSteelBlue
emacs*bitmapIcon: on
!emacs*font: fixed
emacs*font: -misc-fixed-*-*-*-*-15-*-75-75-*-*-iso8859-*
emacs.geometry: 80x28
emacs*BorderColor: DarkSlateGray
emacs*fringe.Background: DarkSlateGray
...
```

in cui si modificano le impostazioni dell'editor `emacs` relative ai colori di sfondo e primo piano, si specifica la geometria iniziale ecc. Si noti come la sintassi preveda una riga in cui si specifica il nome della risorsa terminato da "`:`", cui segue la stringa che assegna il valore. Si noti anche come il nome della risorsa possa contenere il carattere jolly "*" che assume lo stesso significato che avrebbe se si specificasse un pathname.[22] Infine si tenga presente che il carattere "`!`" serve ad introdurre una riga di commento.

Se le risorse permettono di controllare i vari parametri interni relativi all'aspetto grafico delle singole applicazioni, quello che contraddistingue l'interfaccia grafica è la capacità di poter eseguire vari programmi in contemporanea, ciascuno nella sua finestra. Come però ci si potrà rendere conto lanciando una sessione X con un singolo programma (come nell'esempio di `startx` visto in sez. 3.3.3), gli X-client si limitano solo a disegnare il contenuto interno alle rispettive finestre. Il posizionamento delle stesse, il loro spostamento, il passaggio dall'una all'altra, ecc. devono essere invece gestiti a parte e farlo non è compito dei singoli programmi. Per questo per poter usare davvero un'interfaccia grafica è necessario avere almeno un altro programma che esegua questi compiti specifici: quello che viene chiamato un *window manager*, e che assume un po' quello che è il ruolo della shell nella riga di comando (lanciare altri comandi e passare dall'uno all'altro).

Senza un *window manager* infatti si avrebbe la grafica ma non la possibilità di spostare, nascondere, ridimensionare le finestre, o di passare dall'una all'altra. È il *window manager* che si cura di tutti questi compiti: del disegno degli elementi grafici di contorno, della gestione dello spostamento attraverso l'uso del mouse, del ridimensionamento e della selezione delle finestre. Un compito fondamentale infatti, quando si hanno più finestre da gestire, è appunto quello della selezione di quella che viene *messa a fuoco*, quella cioè cui verranno inviati tutti gli eventi in ingresso (ad esempio quello che si scrive sulla tastiera), che è un po' l'equivalente del processo in *foreground* visto in sez. 2.1.2.

Per usare con profitto una sessione grafica tutto quello che serve è un *window manager*, molti di questi infatti gestiscono anche la presenza di un menu o di una barra direttamente sullo schermo (o eventuali altri elementi di contorno), usando i quali è possibile lanciare altre applicazioni o avere informazioni sul sistema. I più comuni, tuttora in ampio uso da parte di coloro che sono abituati alla loro interfaccia e non necessitano di funzionalità più avanzate, sono *Window Maker*, *ICEwm*, *fvwm*, *blackbox*, ecc.

Le interfacce grafiche moderne vanno però al di là della semplice gestione delle finestre arrivando a definire una infrastruttura che consenta operazioni come il cosiddetto *drag and drop*, in cui si possono passare elementi o dati da una finestra ad un altra (e cioè da una applicazione ad un altra). Per fare questo un *window manager* non basta, il passo successivo è allora

[22]cioè applica il valore a tutte le risorse che corrispondono alla stringa in cui nella gerarchia l'asterisco è sostituito da un nome qualunque.

quello dell'uso di un *desktop environment*, cioè di una infrastruttura (completa di meccanismi di intercomunicazione) che definisca non solo la gestione delle finestre, ma tutto un insieme di elementi (come oggetti grafici comuni e meccanismi di intercomunicazione) che porta appunto ad un ambiente grafico integrato, analogo a quello che si trova su altri sistemi operativi come MacOS o Windows.

3.3.5 Programmi per l'accessibilità

Benché non necessariamente legati all'uso dell'interfaccia grafica, ed in certi casi anzi totalmente slegati, come quelli indirizzati all'uso da parte di persone non vedenti, tratteremo in questa ultima sezione i programmi realizzati per rispondere alle esigenze di accessibilità in quanto la gran parte degli stessi fanno parte o sono strettamente legati agli ambienti desktop.

Le tecnologie legate alla accessibilità sono diverse, in quanto sono anche diverse le tipologie di disabilità alle quali cercano di fornire una risposta. Una prima serie di programmi, per lo più integrati con gli ambienti desktop (come GNOME e KDE) sono quelli che consentono di modificare le modalità di utilizzo di tastiera e mouse per le persone con problemi di mobilità.

In entrambi i casi i programmi di configurazione dell'ambiente consentono di modificare il comportamento standard della tastiera per persone con difficoltà motorie, ed in particolare si possono trovare configurazioni specifiche per:

- bloccare la ripetizione dell'immissione di un carattere quando si mantiene un tasto premuto a lungo;
- ignorare pressioni multiple ravvicinate;
- simulare la pressione contemporanea di due tasti usando la permanenza della pressione dei tasti modificatori (*shift*, *alt*, *ctrl*, ecc.);
- modificare la velocità di spostamento del mouse;
- modificare le tempistiche delle pressioni dei tasti del mouse;
- simulare i click con la pressione in un'area;

ed in particolare si possono impostare per ciascuna di queste funzionalità i tempi di risposta ed attesa, in modo da adattarle alle esigenze specifiche di una persona, che possono variare in ragione della diversità delle sue difficoltà motorie.

Inoltre per le persone in grado di usare soltanto un mouse o un dispositivo di puntamento esistono diversi programmi per utilizzare una tastiera virtuale presentata sullo schermo, su cui simulare l'uso della tastiera con la pressione dei tasti effettuata tramite il mouse. Fra questi uno dei più sofisticati è `gok` (acronimo di *GNOME On-screen Keyboard*) che consente una emulazione molto dettagliata di tutte le funzionalità di una tastiera ordinaria, e si integra anche con il Desktop offrendo scorciatoie e "*gestures*" per l'uso del menù e delle barre.

Una seconda serie di programmi sono previsti per coloro che hanno difficoltà visive. Ad esempio tutti gli ambienti Desktop prevedono la possibilità per gli utenti di usare programmi di ingrandimento per lo schermo (ad esempio `kmag` per KDE) e di modificare le dimensioni dei font dei caratteri per aumentare la visibilità. Esistono inoltre specifici temi grafici ad alto contrasto per gli ambienti grafici, per favorire chi ha difficoltà visive nelle differenze di colori o di luce.

A questi programmi si affiancano, rivolti specificamente ai non vedenti, alcuni programmi di sintesi vocale. Fra questi spicca `emacspeak`, sviluppato da un non vedente, inizialmente nato come un lettore di schermo per l'editor `emacs` (con il quale offre una integrazione molto stretta) ma in

grado di trasformare l'editor in una sorta di "desktop audio", effettuando non soltanto la lettura dello schermo, ma anche la restituzione audio delle azioni compiute, la rilettura automatica di quanto scritto e la vocalizzazione dell'inserzione dei caratteri.

Sempre rivolto ai non vedenti è `brltty`, un demone che consente di redirigere l'output di una console su una barra Braille (un dispositivo in grado di generare i caratteri Braille su una superficie), per restituire una rappresentazione tattile dello stesso, che supporta anche la sintesi vocale, lo scorrimento all'indietro e l'uso di più terminali virtuali.

Infine nello sviluppo del desktop GNOME tutta una serie di funzionalità relative alle tecnologie di assistenza per l'accesso delle persone non vedenti sono state integrate tramite ORCA (il nome deriva dal porsi come alternativa al proprietario JAWS) presente nella distribuzione ufficiale di questo ambiente desktop fino dalla versione 2.16. Il programma si appoggia alla interfaccia AT-SPI (*Assistive Technology Service Provider Interface*), un toolkit che consente di fornire servizi di accessibilità alle applicazioni che lo utilizzano.

Con ORCA è possibile abilitare tutta una serie di tecnologie di assistenza che vanno ben oltre le originali funzionalità di *screen reader*, con l'uso della sintesi vocale per le varie azioni su desktop, l'integrazione con le barre Braille, la definizione di effetti sonori e combinazioni di tasti per consentire anche a chi non vede di orientarsi all'interno delle applicazioni.

3.4 Il sistema di stampa

Tratteremo in questa sezione la configurazione delle stampanti sotto GNU/Linux; dopo una introduzione generale sulla gestione generica della stampa vedremo in particolare la configurazione secondo il sistema tradizionale del *Line Printing Daemon* di BSD e la nuova architettura del *Common Unix Printing System*.

3.4.1 Introduzione generale

Mantenendo la filosofia progettuale di Unix per cui *tutto è un file*, è facile immaginarsi che la prima e più immediata modalità di stampa sia quella di inviare il file da stampare direttamente al dispositivo associato alla stampante. Questo è senz'altro possibile, ma si scontra con una serie di problemi che lo rendono altamente sconsigliabile; il primo è che non esiste un dispositivo univoco che identifichi una stampante, questa può essere infatti agganciata ai dispositivi più vari (e molte stampanti ne supportano più di uno), come la parallela, la seriale, una porta USB, ecc. In ciascuno di questi casi il file andrebbe inviato sul relativo dispositivo (ad esempio nel caso di stampante su porta parallela, a `/dev/lp0`).

Tutto questo non costituirebbe di per sé un grosso problema, basterebbe identificare il dispositivo specifico indicandolo tramite un link simbolico con un nome generico che vi faccia riferimento. Il problema più importante è invece quello relativo al contenuto del file da stampare; un tempo quando si stampavano solo file di testo senza formattazioni il problema non c'era, oggi però si stampano file con contenuti grafici e con formattazioni complesse e font diversi.

Per questo in genere le stampanti moderne prevedono un loro linguaggio di stampa come il PostScript (alcune ne supportano anche più di uno), che permette di ottenere i risultati voluti. Questo significa che anche un semplice file di testo deve essere opportunamente *trattato* prima di essere inviato alla stampante. Inoltre se, come accade spesso, i programmi generano le stampe

in un formato (principalmente il PostScript) e la stampante non lo capisce, questo dovrà essere opportunamente tradotto.

Il terzo problema è quello di essere in presenza di un sistema operativo multiutente e multitasking; il che comporta che con l'accesso diretto si possono avere più processi che scrivono in contemporanea sul dispositivo della stampante, coi relativi dati che si mescolano fra di loro dando luogo a risultati tutt'altro che piacevoli nella stampa finale.

Per questa serie di motivi in genere il dispositivo della stampante non è mai accessibile direttamente ai singoli utenti, ma viene gestito attraverso un opportuno *sistema di stampa* che si incarica di risolvere questi problemi. In generale quello che succede è che l'accesso alla stampante è regolato attraverso un opportuno demone, che è l'unico che ci scrive, le stampe vengono richieste inviando i file da stampare al suddetto demone, che si incarica di eseguire tutte le operazioni necessarie (comprese le eventuali conversioni di formato) e di garantire un accesso corretto senza sovrapposizioni.

Una delle caratteristiche fondamentali dei sistemi di stampa in ambiente unix-like è quella delle *code*, le stampe cioè non vengono mai eseguite direttamente, ma immesse, tramite un opportuno comando, su una coda di stampa dalla quale poi verranno prelevate (in genere dal demone di cui sopra) ed opportunamente inviate sulla stampante secondo lo schema illustrato in fig. 3.2.

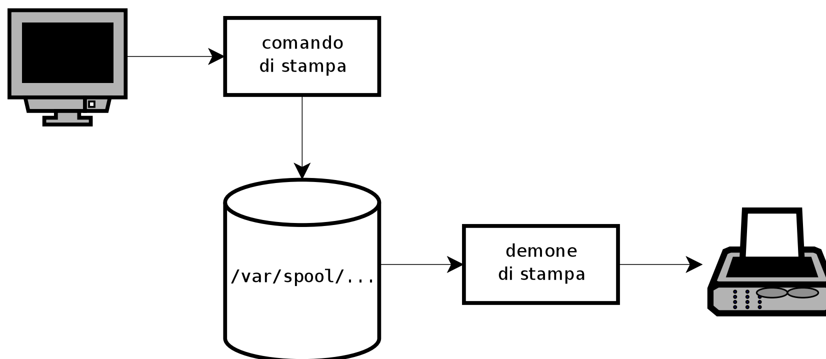


Figura 3.2: Schema generico del funzionamento del sistema di stampa.

Il vantaggio di questo sistema è che la gestione della coda può essere eseguita indipendentemente dalla gestione della stampante, ad esempio diventa possibile inviare la stampa su una coda remota se è previsto un opportuno meccanismo di comunicazione. Lo stesso concetto di stampa remota può essere realizzato in maniera alternativa usando una coda locale ed usando direttamente il demone di stampa per inviare i file ad un altro demone remoto.

Inoltre con lo schema di fig. 3.2 nell'eseguire l'invio dei file dalla coda di stampa alla stampante il demone di stampa può eseguire tutte le operazioni necessarie (realizzate in genere attraverso quelli che si chiamano *filtri di stampa*) per convertire il contenuto del file da stampare in un formato che la stampante è in grado di comprendere.

Infine l'uso di questa architettura permette anche di utilizzare più code di stampa per la stessa stampante; questo consente ad esempio di utilizzare code diverse (con filtri diversi) a seconda del

tipo di file che si vuole stampare, o con diversi privilegi o priorità. Benché la terminologia usata finora faccia riferimento all'architettura classica ereditata da BSD (che vedremo sommariamente in sez. 3.4.4) i concetti e la filosofia generale di funzionamento restano gli stessi anche con la più recente architettura di CUPS.

3.4.2 Il *Common Unix Printing System*

Benché continuino ad esistere altri programmi per la gestione del sistema di stampa (torneremo sulla compatibilità con il vecchio sistema di BSD in sez. 3.4.4) oggi questa viene effettuata quasi esclusivamente tramite CUPS (acronimo di *Common Unix Printing System*).

A differenza dei programmi usati nelle prime versioni di Unix, nati in un periodo in cui le stampanti erano sempre collegate fisicamente alle singole macchine e venivano utilizzate per lo più solo localmente, CUPS è stato creato nell'era di Internet, con stampanti che possono operare direttamente sulla rete, e per questo fornisce anche tutta l'interfaccia di gestione via rete,[23] tramite apposito servizio con cui si comunica usando una variante del protocollo HTTP su una porta dedicata (il default è la 631). Rispetto al protocollo HTTP ordinario un server CUPS prevede però una comunicazione con caratteristiche specifiche, che sono state standardizzate nell'RFC 3239 in cui è definito il protocollo IPP (acronimo di *Internet Printing Protocol*).

L'uso di un protocollo di rete che è basato su HTTP ha il grande vantaggio che con CUPS buona parte dell'amministrazione del sistema di stampa, come la creazione il controllo e la rimozione di stampe e code di stampa o la configurazione delle stampanti, può essere eseguita direttamente tramite un browser qualunque, che basterà puntare sulla porta 631 della macchina da amministrare. Questa resta la modalità principale di utilizzo del sistema, anche se è possibile usare degli opportuni comandi di shell per eseguire gli stessi compiti.

Il servizio è fornito da un demone, `cupsd`, controllato da un file di configurazione, `cupsd.conf`, che di norma viene installato in `/etc/cups`. Questo file configura il servizio nel suo insieme, dato che come accennato la gestione di stampanti, code e stampe avviene attraverso l'interfaccia che il servizio stesso mette a disposizione. Il formato del file è simile a quello del file di configurazione del web server Apache (il server web più diffuso), la maggior parte sono una serie di direttive elementari nella forma:

```
NomeDirettiva argomento
```

dove il nome della direttiva è composto da normali caratteri, e il successivo argomento viene specificato di seguito come valore separato da uno o più spazi il cui formato dipende dalla direttiva stessa. Al solito le righe vuote o inizianti per un "`#`" vengono ignorate.

Ad esempio con la direttiva `Port` si può specificare la porta su cui il servizio si pone in ascolto (il default è la 631), mentre con la direttiva `Listen` si può indicare esplicitamente un indirizzo IP o anche un socket locale. Inoltre una delle caratteristiche più interessanti di CUPS è la sua capacità di riconoscere automaticamente le stampanti presenti su una rete, con il cosiddetto *browsing*.

Questa funzionalità è attivata di default (ma può essere disattivata impostando ad `off` la direttiva `Browsing`) e può essere controllata con la direttiva `BrowseAddress` per indicare l'indirizzo

[23] i concetti relativi a indirizzi, porte, protocolli, ed in generale tutto quanto riguarda gli argomenti relativi alla rete citati in questa sezione vengono assunti come noti, per la trattazione degli stessi si rimanda a quanto verrà illustrato nel capitolo 7.

di *broadcast* (vedi sez. 7.2.2) cui inviare le richieste di riconoscimento. Normalmente tutte le richieste vengono accettate, ma con le direttive `BrowseAllow` e `BrowseDeny` si possono indicare quali sono le macchine (o le reti) le cui richieste vengono accettate o rifiutate.

Direttiva	**Significato**
`AccessLog`	il file su cui vengono registrati gli accessi; richiede come valore il nome del file o la parola chiave `syslog` che invia le informazioni all'omonimo servizio.
`BrowseAddress`	indirizzo di `broadcast` cui inviare le richieste di riconoscimento.
`BrowseAllow`	indirizzo o rete da cui accettare le richieste di riconoscimento.
`BrowseDeny`	indirizzo o rete da cui non accettare le richieste di riconoscimento.
`Browsing`	abilita o disabilita il riconoscimento automatico delle stampanti presenti (prende i valori `on` o `off`).
`DataDir`	la directory dove sono mantenuti i dati di CUPS.
`DocumentRoot`	la directory radice per i documenti HTTP (come la documentazione) che vengono messi a disposizione attraverso l'interfaccia.
`ErrorLog`	il file su cui vengono registrati gli errori; richiede come valore il nome del file o la parola chiave `syslog` che invia le informazioni all'omonimo servizio.
`Group`	gruppo per conto del quale viene eseguito `cupsd` (di default `lpadmin`).
`Include`	permette di includere un altro file nella configurazione.
`Listen`	consente di specificare completamente l'indirizzo IP su cui il server si pone in ascolto nella forma `address:port` (o `/path/name` in caso di socket locale).
`LogLevel`	livello dei messaggi da parte del demone.
`PageLog`	il file su cui vengono registrati i dati sulle pagine stampate; richiede come valore il nome del file o la parola chiave `syslog` che invia le informazioni all'omonimo servizio.
`Port`	specifica la porta su cui si pone in ascolto il demone `cupsd` (il default è 631).
`Printcap`	specifica su quale file riprodurre un file `printcap` che faccia da sostituto per `/etc/printcap` per i programmi che necessitano di quest'ultimo per riconoscere le code di stampa.
`ServerRoot`	specifica la directory radice per il demone `cupsd`, di default è `/etc/cups`.
`SystemGroup`	specifica il gruppo (o i gruppi, se si indica un elenco separato da spazi) i cui utenti avranno i privilegi amministrativi sul sistema (su Debian viene usato `lpadmin`).
`TempDir`	la directory in cui inserire i file temporanei, in sostanza lo *spool* di stampa (di default è `/var/spool/cups/tmp`).
`User`	utente per conto del quale viene eseguito `cupsd` (di default `lp`).
`MaxCopies`	numero massimo di copie che un utente può richiedere, il valore 0 significa nessun limite, il default è 100.
`MaxJobs`	numero massimo di stampe in coda, il valore 0 significa nessun limite, il default è 500.
`MaxJobsPerUser`	numero massimo di stampe per utente, il default è 0 e significa nessun limite.

Tabella 3.14: Principali direttive per il file `/etc/cups/cupsd.conf`.

In tab. 3.14 sono riportate le direttive di base più importanti di `/etc/cups/cupsd.conf`, per l'elenco completo con una descrizione sommaria si può fare riferimento alla pagina di manuale. Inoltre le distribuzioni normalmente installano una versione predefinita di questo file in cui tutte le opzioni (anche quelle non attivate esplicitamente) sono abbondantemente commentate. Per una documentazione completa invece si può fare riferimento ai manuali on-line direttamente accessibili attraverso l'interfaccia del sistema (un altro dei vantaggi di avere un protocollo basato su HTTP) o reperibili su `http://www.cups.org/`.

Oltre alle direttive appena illustrate, che usano la sintassi elementare in cui si esprime un argomento con un valore ad una parola chiave, il formato di `cupsd.conf` prevede la presenza

di direttive con una sintassi più complessa, che vengono chiamate *contenitori* (o *container*) in quanto in genere consentono al loro interno altre direttive.

In particolare sono queste le direttive usate per effettuare il controllo degli accessi all'interfaccia di rete fornita da `cupsd`, con cui è possibile gestire il controllo del sistema di stampa attraverso il protocollo IPP. Queste direttive, che in genere vengono raggruppate in coda al file (ma possono essere inserite in qualunque posizione), hanno una forma diversa:

```
<DirettivaContenitore argomento>
  DirettivaNormale1 argomento1
  DirettivaNormale2 argomento2
  ...
</DirettivaContenitore>
```

La più comune di queste direttive è `Location`, nel qual caso il relativo argomento assume sempre l'aspetto di un pathname, o più precisamente una URI.[24] Questa direttiva viene in genere utilizzata per il controllo di accesso, poiché l'accesso alle varie funzionalità del sistema di stampa via IPP avviene tramite indirizzi web, e viene gerarchizzato in una struttura ad albero analoga a quella dei file, ed espressa appunto con dei "*percorsi*" a partire da una radice "`/`" avendo accesso alla quale si diventa in grado di compiere qualunque operazione.

Specificando un percorso a partire dalla radice si potrà accedere ad una classe di operazioni, sempre più specifica via via che si scende nella gerarchia, per la quale si potranno impostare le regole di accesso. I principali percorsi utilizzabili come argomento di `Location` sono illustrati in tab. 3.15, insieme alle operazioni cui si accede attraverso di essi; per l'elenco completo si può fare riferimento alla documentazione in linea di CUPS.

Percorso	**Descrizione**
`/`	accesso a tutte le operazioni.
`/admin`	accesso operazioni amministrative (creazione, cancellazione, impostazione e avvio delle stampanti e delle code).
`/admin/conf`	accesso ai file di configurazione.
`/classes`	accesso alle operazioni sulle classi.
`/classes/name`	accesso alle operazioni sulla classe `name`.
`/jobs`	accesso alle operazioni sulle stampe.
`/printers`	accesso alle operazioni sulle stampanti.
`/printers/name`	accesso alle operazioni sulla stampante `name`.

Tabella 3.15: I *percorsi* per l'accesso alla gestione delle operazioni usabili come argomento di `Location`.

Una volta che con la scelta di un percorso nella direttiva container `Location` si sia definito a quale classe di operazioni si vuole che siano applicate le regole di accesso, queste ultime devono essere specificate da altrettante direttive poste all'interno del "*contenitore*". Questo viene fatto attraverso una serie di ulteriori direttive, specifiche per l'indicazione di regole di accesso, che hanno una sintassi identica alle direttive elementari viste in precedenza, ma che sono utilizzabili solo all'interno del "*contenitore*".

Una direttiva `Location` prevede sempre la presenza di una direttiva `Order` che definisce la politica di accesso, che per chiarezza si pone in genere come prima direttiva al suo interno. Se

[24] per una trattazione del significato delle URI e delle URL si rimanda alla sezione 1.1.2 di [WebServ].

l'accesso è consentito per default si utilizza il valore `Deny,Allow`, mentre se è vietato per default si utilizza il valore `Allow,Deny`. Alla direttiva `Order` devono poi seguire successive direttive `Deny` o `Allow` per indicare rispettivamente le macchine a cui vietare o consentire esplicitamente gli accessi, e direttive come `AuthType` e `AuthClass` per indicare le eventuali modalità di autenticazione. I valori utilizzabili per queste direttive sono riportati in tab. 3.16, in cui sono illustrate le principali direttive di accesso, per una descrizione più dettagliata di tutte le direttive e della relativa sintassi si può fare nuovamente riferimento alla documentazione di CUPS.

Direttiva	**Significato**
`Order`	definisce la modalità con cui vengono gestite le autorizzazioni con le direttive `Deny` e `Allow`, prende come valori: `Deny,Allow` in cui si accettano tutte le richieste eccetto quelle negate da una direttiva `Deny` e `Allow,Deny` in cui si accettano solo le richieste consentite con una direttiva `Allow`.
`AuthType`	definisce le modalità di autenticazione, prende come valori: `None` che non effettua nessuna autenticazione, `Basic` che effettua l'autenticazione usando le credenziali standard della macchina, `Digest` e `BasicDigest` che utilizza le credenziali specifiche memorizzate in `/etc/cups/passwd.md5`.
`Allow`	consente l'accesso: prende come valore l'indirizzo IP di una singola macchina o di una rete (in notazione CIDR, vedi sez. 7.2.2) o un nome a dominio (consentendo notazioni del tipo `*.dominio.it` o `.dominio.it` per indicare tutti le macchine in un dominio) o la parola chiave `All`.[25]
`Deny`	nega l'accesso, usa gli stessi valori appena descritti per `Allow`.
`Encryption`	abilita la cifratura della connessione, prende come valori: `Never`, `IfRequested` e `Required`, il cui significato è autoesplicativo.
`Require`	richiede un accesso autenticato (che verrà effettuato secondo la modalità indicata da `AuthType`) consentendo di specificare i soggetti che possono usare tale accesso, sia in forma di lista di utenti, da far seguire alla parola chiave `user`, sia indicando un gruppo, con la parola chiave `group`; il gruppo può essere indicato indicato anche con `user` con la sintassi `@`*`nome`* che si può utilizzare anche per indicare i valori predefiniti `@OWNER`, che indica il proprietario di un lavoro di stampa, e `@SYSTEM` che indica il gruppo impostato dalla direttiva `SystemGroup`.

Tabella 3.16: Le direttive di accesso usate in all'interno di una `Location`.

Un esempio del contenuto di `/etc/cups/cupsd.conf`, estratto (togliendo i commenti) dalla versione installata con il pacchetto `cupsys` su una Debian Squeeze, è il seguente:

/etc/cups/cupsd.conf

```
LogLevel warn
MaxLogSize 0
SystemGroup lpadmin
Listen localhost:631
Listen /var/run/cups/cups.sock
Browsing On
BrowseOrder allow,deny
BrowseAllow all
BrowseLocalProtocols CUPS dnssd
DefaultAuthType Basic
<Location />
  Order allow,deny
</Location>
<Location /admin>
  Order allow,deny
</Location>
```

```
<Location /admin/conf>
  AuthType Default
  Require user @SYSTEM
  Order allow,deny
</Location>
...
```

Si noti come nella configurazione di `cupsd.conf` non vi sia nessuna traccia di dati relativi alle stampanti. Queste infatti sono mantenute in un file a parte, `printers.conf`, ma in genere il contenuto di questo file viene gestito da `cupsd`, che ne genera e modifica il contenuto sia attraverso i comandi che con l'interfaccia di gestione via web. Questo comporta anche che il file non deve essere modificato manualmente fintanto che il programma è attivo.

Il formato del file mantiene le caratteristiche di base di `cupsd.conf`, ma in questo caso le direttive sono diverse. La principale è la direttiva container `Printer` che definisce una stampante logica, al cui interno vanno poi inserite le direttive che ne controllano l'uso.

Oltre alla configurazione lato server è possibile specificare una configurazione generale sul lato client (per tutti i programmi cioè che si appoggiano a CUPS per la stampa), che viene mantenuta a livello di sistema nel file `/etc/cups/client.conf`, e può essere personalizzata dal singolo utente in un file `.cupsrc` nella propria home directory.

In genere le direttive principali che si utilizzano in detti file sono `ServerName` che prende come valore il nome della macchina cui rivolgersi come server di stampa e `Encryption` che prende gli stessi valori della omonima vista in tab. 3.16 per il server, che indica la modalità con cui eseguire una eventuale cifratura della connessione.

3.4.3 I comandi di gestione per CUPS

Come accennato anche se la gran parte della gestione e della configurazione del sistema di stampa viene gestita attraverso l'interfaccia web fornita da CUPS è comunque possibile utilizzare anche una serie di comandi di shell; il primo di questi è `lpadmin`, che è quello che si può utilizzare per definire le proprietà generali delle stampanti presenti nel sistema.

Il concetto di base è quello di definire delle *stampanti logiche*, chiamate così in quanto se ne possono associare più di una ad una stessa stampante fisica. Si tratta in sostanza della reimplementazione del concetto di coda di stampa visto in sez. 3.4.1. Alle stampanti logiche si deve fare sempre riferimento con l'opzione `-p` seguita dal nome. In pratica tutte le invocazioni di `lpadmin` richiedono l'uso di questa opzione, tranne il caso in cui si usi al suo posto `-x` che richiede la cancellazione di una stampante logica.

Per creare una stampante logica oltre ad identificarla con `-p` occorrerà anche definire con quale dispositivo vi si accede, cosa che si fa attraverso l'opzione `-v`; questa prende come argomento quello che la documentazione chiama un *device-URI*, cioè una stringa nella forma di un indirizzo simile a quello di un sito web, che indica con quale modalità è possibile raggiungere la stampante stessa (se sulla porta parallela, seriale, USB, o via rete secondo vari protocolli di comunicazione), un breve elenco di possibili valori è riportato in tab. 3.17.

Oltre a definire come si può parlare con la stampante è necessario anche specificare quale "*linguaggio*" usare, questo si fa con l'opzione `-P` che associa alla stampante logica il relativo file PPD (*PostScript Printer Description*) che ne descrive le capacità; questo è un po' l'equivalente

Nome	Descrizione
`ipp://remote.printer.ip`	stampante di rete raggiungibile con protocollo IPP.
`smb://remote.printer.ip`	stampante di rete windows raggiungibile tramite Samba.
`lpd://remote.printer.ip`	stampante di rete raggiungibile con protocollo LPD.
`parallel:/dev/lp0`	stampante locale sulla porta parallela.
`usb:/dev/lp0`	stampante locale su porta USB.
`serial:/dev/ttyS0`	stampante locale su porta seriale.

Tabella 3.17: Le diverse possibili specificazioni per gli indirizzi del dispositivo associato ad una stampante (*device-URI*).

del filtro di stampa, in quanto serve a definire quali sono le modalità con cui alla fine i file vengono stampati.

Una delle caratteristiche più interessanti di CUPS è quella di permettere di definire delle *classi* di stampanti, in sostanza una classe definisce un gruppo di stampanti che può essere trattato come un insieme, così che indicando di stampare su quella classe la stampa viene effettuata su una delle stampanti che ne fanno parte. La definizione delle classi viene fatta tramite le opzioni `-c` e `-r` di `lpadmin` che permettono rispettivamente di inserire o eliminare una stampante dalla classe passata come argomento; se una classe non esiste viene creata all'inserimento della prima stampante, se si elimina l'ultima stampante la classe viene cancellata.

Opzione	Descrizione
`-p` *stampante*	indica la stampante logica su cui si sta operando.
`-x` *stampante*	rimuove una stampante logica.
`-d` *stampante*	imposta la stampante come stampante di default.
`-c` *classe*	inserisce in una classe una stampante logica.
`-r` *classe*	rimuove una stampante logica da una classe.
`-v` *URI*	indica il dispositivo da usare per parlare con la stampante (secondo il formato illustrato in tab. 3.17).
`-P` *file.ppd*	definisce il file di descrizione della stampante da utilizzare.
`-E`	abilita una stampante rendendola disponibile per la stampa.
`-h` *host*	indica il server CUPS remoto su cui far operare il comando.

Tabella 3.18: Principali opzioni del comando `lpadmin`.

Le principali opzioni del comando sono riportate in tab. 3.18; al solito si può fare riferimento alla pagina di manuale o alla documentazione di CUPS per un elenco completo e per la descrizione dettagliata.

Una volta definita una stampante logica questa può essere abilitata o disabilitata alla stampa rispettivamente con i comandi `cupsenable` e `cupsdisable` che prendono come argomento il nome della stessa; entrambi i comandi prevedono l'uso dell'opzione `-h` per indicare un server remoto. Inoltre `cupsdisable` supporta l'opzione `-c` che permette di rimuovere tutte le stampe pendenti dalla stampante disabilitata, e `-r` che prende come argomento una stringa in cui descrivere le motivazioni per cui la stampante è stata disabilitata (che sarà riportata dai comandi diagnostici).

Si tenga presente comunque che abilitare o disabilitare una stampante logica significa sostanzialmente abilitare o disabilitare l'esecuzione delle stampe su di essa, e che questa è una operazione diversa rispetto a quella di accettare o rifiutare l'immissione di nuove stampe sulla relativa coda; questa seconda operazione viene fatta invece con i comandi `cupsaccept` e `cupsreject`,

che di nuovo richiedono come argomento il nome della stampante; come per `cupsdisable` anche `cupsreject` supporta l'opzione `-r` con lo stesso significato.

Il comando per richiedere una stampa è invece `lp`,[26] che prende come argomento il file da stampare, e lo invia sulla stampante di default, riportando a video l'identificatore associato alla stampa richiesta. Le stampe infatti sono chiamate, seguendo la nomenclatura inglese, *job*, in particolare i comandi di gestione fanno riferimento a questi identificativi come ai *job ID* e riportano il numero di stampe correnti come numero di *job*.

Qualora si debba indicare una stampante specifica al posto di quella di default si deve usare l'opzione `-d` che prende come parametro il nome della stampante o della classe cui inviare il lavoro di stampa, mentre se si vuole stampare utilizzando uno specifico server CUPS remoto occorre specificarne l'indirizzo come parametro dell'opzione `-h` (nella forma `host:porta`).

Questa, insieme alle altre opzioni riportate nella prima parte di tab. 3.19) è una delle opzioni generiche che controllano la modalità di collegamento al server che sono disponibili in generale per tutti i comandi client, compresi i precedenti comandi di controllo delle code, ed i comandi per le statistiche e le informazioni che vedremo più avanti.

Opzione	**Descrizione**
`-E`	forza una comunicazione cifrata col server.
`-h` *`host`*	specifica il server CUPS remoto cui inviare la stampa.
`-U` *`user`*	indica un utente con cui collegarsi al server.
`-d` *`dest`*	specifica la stampante o la classe da usare.
`-i` *`job`*	specifica la stampa su cui operare, indicata per *job ID*.
`-n` *`N`*	specifica il numero di copie da stampare.
`-o` *`opts`*	passa una o più opzioni per la stampante, indicate come una stringa contenente una lista di assegnazioni nella forma `nome=valore` separate da spazi (i valori dipendono dalla stampante specifica).
`-q` *`N`*	specifica una priorità (valore numerico fra 1 e 100).
`-H` *`when`*	specifica quando eseguire una stampa: `hold` la blocca indefinitamente, un tempo in formato `HH:MM` la blocca ad allora, `resume` la avvia immediatamente e `restart` riesegue una stampa completata (gli ultimi due valori richiedono l'uso di `-i` per identificare la stampa cui si fa riferimento).
`-P` *`pages`*	specifica quali pagine stampare (passati come lista separate da virgole o intervalli separati da un "-").

Tabella 3.19: Principali opzioni del comando `lp`.

Altre due opzioni utili sono `-n`, che permette di specificare il numero di copie da stampare e `-q`, che permette di impostare una priorità (un numero da 1 a 100, con valore di default pari a 50). Le opzioni principali sono riportate in tab. 3.19, per l'elenco completo ed i dettagli si può fare riferimento alla pagina di manuale.

Per cancellare una stampa si può invece usare il comando `cancel`, che prende come argomento l'identificatore della stampa, o, se specificato con l'opzione `-a`, una stampante o una classe, nel qual caso saranno cancellate tutte le stampe presenti su di essa. Il comando supporta anche l'opzione `-h` con lo stesso significato di `lp`; al solito per i dettagli si faccia riferimento alla pagina di manuale.

[26] anche se CUPS supporta un sistema di compatibilità all'indietro con il protocollo tradizionale di BSD che consente l'uso del comando `lpr`, con la sintassi illustrata in sez. 3.4.4.

Infine per ottenere informazioni riguardo il sistema di stampa ci sono alcuni comandi ulteriori. Il primo è `lpinfo` che permette di vedere i dispositivi fisici ed i supporti per le stampanti disponibili all'interno di CUPS. Il comando richiede una delle opzioni `-v` o `-m` per stampare rispettivamente la lista dei dispositivi utilizzabili e quella delle stampanti supportate. Ad esempio se si vuole ottenere una lista dei dispositivi visibili da riutilizzare con l'opzione `-v` di `lpadmin` si può eseguire il comando:

```
hain:~# lpinfo -v
network http
network ipp
network lpd
direct parallel:/dev/lp0
direct usb:/dev/usb/lp0
...
direct usb:/dev/usb/lp15
network smb
```

Oltre a quelle indicate il comando prevede le opzioni per il collegamento verso il server illustrate nella prima parte di tab. 3.19, ed una serie di altre opzioni per selezionare in maniera più dettagliata i risultati riportati, per le quali al solito si rimanda alla consultazione della pagina di manuale.

Il secondo comando è `lpstat`, che riporta le informazioni relative alle singole stampanti, classi e stampe attive. Anche questo comando supporta le opzioni per il collegamento illustrate nella prima parte di tab. 3.19.

Opzione	**Descrizione**
`-c` *[class]*	elenca le classi e le stampanti che ne fanno parte, limitandosi alla eventuale classe indicata.
`-p` *[dest]*	riporta lo stato di una stampante, eventualmente limitandosi a quella indicata.
`-a` *[queue]*	riporta lo stato di una coda, eventualmente limitandosi a quella indicata.
`-o` *[dest]*	riporta l'elenco dei lavori di stampa, limitandosi alla eventuale destinazione indicata.
`-W` *jobs*	specifica quali lavori di stampa riportare negli elenchi, prevede tre valori di significato evidente: `all`, `completed` e `non-completed` (il default).
`-l`	esegue un elenco generico dei dati di stampanti, classi e lavori di stampa.
`-d`	indica la attuale stampante di default.

Tabella 3.20: Principali opzioni specifiche del comando `lpstat`.

L'opzione `-c` richiede la stampa delle informazioni relative ad una classe (o a tutte se non si specifica quale classe), l'opzione `-a` riporta lo stato di una coda (e delle eventuali stampe in attesa), mentre l'opzione `-p` riporta lo stato di una stampante; entrambe richiedono il nome di una stampante logica da interrogare, e riportano lo stato di tutte quelle definite se questa non viene specificata. Si sono riportate in tab. 3.20 le opzioni principali del comando. Ulteriori dettagli e l'elenco completo delle opzioni si trovano al solito sulla pagina di manuale.

Il terzo comando è `lpoptions` che permette di consultare e modificare opzioni e default relativi alle stampanti. Come i precedenti riconosce le opzioni di collegamento illustrate nella

prima parte di tab. 3.19, se usato con la sola opzione `-l` stampa le opzioni relative alla stampante di default, ma se ne può scegliere una qualunque specificando il relativo nome con l'opzione `-p`. Indicando un nome con l'opzione `-d` si imposta invece una nuova stampante di default.

Lo scopo principale del comando è però impostare le opzioni delle stampanti, i cui valori dipendono dalle singole stampanti e che non è possibile elencare qui. L'impostazione di un valore si può fare con l'opzione `-o` con cui si possono inserire o modificare i valori delle opzioni nella forma `opzione=valore`, mentre con `-r` seguito dal nome dell'opzione si può rimuovere quest'ultima dalla configurazione della stampante.

3.4.4 Il sistema di stampa in stile BSD

Benché oggi non sia praticamente più usato, fino a qualche anno fa, prima dell'avvento di CUPS, il sistema di stampa più diffuso in ambiente Unix era quello sviluppato per BSD, e tutt'ora distribuito insieme a questo sistema operativo. Di questo è disponibile una versione anche GNU/Linux, ma il suo funzionamento è stato a lungo difettoso ed il numero di stampanti supportate è limitato. Per questo esso veniva in genere sostituito da una reimplementazione alternativa sviluppata indipendentemente, LPRng, che pur mantenendo la stessa interfaccia e le stesse funzionalità, supportava diverse estensioni e molte più stampanti.

Oggi nessuna delle due varianti viene più utilizzata per gestire le stampanti, essendo entrambe nettamente inferiori, sia in termini di funzionalità che di stampanti utilizzabili, a CUPS. Esiste però, per supportare i programmi che ancora fanno riferimento alla vecchia interfaccia, un layer di compatibilità fornito dallo stesso CUPS,[27] che consente di implementare la vecchia interfaccia del protocollo LPD (acronimo di *Line Printer Daemon*) standardizzata nell'RFC 1179.

Il cuore del sistema di stampa in stile BSD è infatti costituito dal demone di stampa `lpd` (da cui deriva il nome del protocollo) che si incarica sia della gestione delle code che dell'invio dei dati alle stampanti. Il protocollo LPD prevede che il demone possa essere contattato anche via rete (da un altro demone o direttamente da un client) per eseguire la stampa in remoto. Nel caso di CUPS il demone viene sostituito da un apposito programma di emulazione, `cups-lpd` che viene normalmente lanciato a richiesta da programmi come `inetd` o `xinetd` (vedi sez. 8.1.2 e sez. 8.1.3) per ricevere le connessioni dai client e reinviare i dati a CUPS.

Oltre al server `lpd` il sistema di stampa in stile BSD prevede poi una serie di programmi client usati per l'immissione delle stampe nelle code e per la gestione di queste ultime. Anche questi sono stati rimpiazzati da opportuni sostituti dal funzionamento identico, ma che inviano invece la stampa ad un server CUPS, e sono questi quelli che tratteremo dato che sono gli unici che ancora hanno una applicazione pratica in quanto gli script ed i programmi scritti quando ancora CUPS non era in auge possono fare ricorso ad essi.

Il più rilevante di questi programmi è `lpr`, con cui si può richiedere la stampa di un file. Il comando prende come argomenti i file da stampare e se non si specifica nulla viene usato lo *standard input*. Una volta messa in coda una serie di stampe il comando riporta sullo *standard output* un numero identificativo per ciascuna di esse che potrà essere usato con gli altri comandi per fare riferimento alle stesse.

L'opzione principale di `lpr` è `-P` che permette di indicare su quale coda di stampa inviare il file da stampare. Se non la si specifica viene usato, se definito, il valore di una delle variabili `PRINTER`,

[27] nel caso di Debian questo viene fornito dal pacchetto `cups-bsd` (o `cupsys-bsd` per le versioni più vecchie).

Opzione	**Significato**
-P *dest*	indica la coda su cui stampare.
-h	inibisce la stampa di una pagina di intestazione.
-# *N*	specifica il numero di copie da stampare.
-J *nome*	associa alla stampa un nome in modo che questo venga stampato nell'intestazione (quando questa è prevista).
-m *mail*	richiede l'invio di una email all'indirizzo specificato qualora la stampa fallisca.
-U *user*	permette di specificare l'utente per conto del quale eseguire la stampa (utilizzabile solo dall'amministratore).

Tabella 3.21: Le principali opzioni del comando `lpr`.

`LPDEST` o la stampante di default di sistema o quella indicata per l'utente con `lpoptions`. Un'altra opzione utile è `-h` che inibisce la stampa di eventuali pagine iniziali per le code in cui questa è prevista e `-#` che richiede la stampa di un certo numero di copie; le altre opzioni principali sono riportate in tab. 3.21, per l'elenco completo al solito si faccia riferimento alla pagina di manuale.

Per la gestione delle stampe sono disponibili una serie di comandi per controllare lo stato della coda ed eseguirvi delle operazioni, il primo di questi è `lpq`, che permette di esaminare lo stato della coda. Se invocato senza argomenti esso stampa la lista delle stampe presenti, con il formato:

```
parker:/root# lpq
Printer: rlp@parker 'Remote printer entry' (dest epson@davis)
 Queue: no printable jobs in queue
 Server: no server active
 Status: job 'root@parker+648' saved at 16:48:21.800
 Rank   Owner/ID                  Class Job Files               Size Time
done   root@parker+648              A   648 /root/iptables        2748 16:48:21
epson is ready and printing
root: active                            [job 543 localhost]
        /root/iptables                          3072 bytes
```

altrimenti si può specificare come argomento il numero identificativo della stampa (quello riportato nella colonna `Job`) ed il comando riporterà solo le informazioni ad essa relative.

Di default il comando esegue la ricerca delle stampe sulla coda di default, a meno di non usare l'opzione `-P` per selezionare una coda specifica. Si può usare l'opzione `-a` per fare eseguire la ricerca su tutte le code. Per le altre opzioni e per una descrizione dettagliata si può fare riferimento alla pagina di manuale.

Una volta creata una stampa, la si potrà eliminare dalla coda con il comando `lprm`. Questo prende come argomenti la lista delle stampe da rimuovere che possono essere indicate tramite il loro identificativo (quello stampato da `lpr` o riportato da `lpq`), mentre l'uso dell'argomento "`-`" seleziona tutte le stampe presenti, se invece non si specifica nulla viene selezionata la prima che può essere rimossa. Di nuovo il comando opera sulla coda di default a meno di non usare l'opzione `-P` per specificarne un'altra o l'opzione `-a` per indicare di operare su tutte le code.

L'uso di `lprm` serve a richiedere la cancellazione delle stampe selezionate, ma un utente potrà rimuovere solo quelle per i quali ha il permesso; con l'opzione `-U` si può richiedere, se si è l'amministratore, di operare a nome di un altro utente. Per le altre opzioni ed i dettagli di funzionamento del programma si faccia riferimento alla pagina di manuale.

Capitolo 4

Amministrazione ordinaria del sistema

4.1 Archiviazione e backup

Una delle attività fondamentali della amministrazione di un sistema è quella dei backup. In questa sezione, dopo una introduzione generale sugli scopi ed i criteri da usare per effettuare i backup, prenderemo in esame i programmi per la gestione degli archivi di dati per la gestione dei backup e la duplicazione dei dati dei dispositivi. Ci limiteremo comunque ai programmi di base che si possono impiegare per eseguire i backup di una singola macchina, per esigenze più complesse esistono programmi dedicati alla esecuzione di backup via rete di interi gruppi di macchine, come Bacula, Amanda o BackupPC. Tratteremo infine le modalità con cui si possono eseguire i backup sia di file e directory che di interi filesystem.

4.1.1 Criteri generali per il backup

Quella dei backup è probabilmente la più importante responsabilità di ogni sistemista. Nel mondo dell'informatica i dati sono il principale prodotto del lavoro svolto, e la perdita degli stessi significa spesso la perdita di grandi quantità di ore di lavoro. Per questo motivo occorre prendere le opportune misure per assicurarsi contro questa evenienza, ed il backup è la principale di queste misure.

Ci possono essere varie ragioni per cui si possono perdere i propri dati, in generale si applica una classificazione che tende a dividerle in quattro categorie principali: disastri naturali, attività umane, errori del software e guasti dell'hardware. Ciascuna di queste categorie comporta rischi specifici e richiede opportune contromisure.

Oggi l'hardware tende ad essere sufficientemente affidabile, ciò non di meno i guasti continuano ad esserci, e la sfortuna è in agguato per farli avvenire nel peggior momento possibile. In genere in questo caso i dispositivi più critici sono i dischi, che per quanto ben costruiti hanno una loro fragilità meccanica e sono comunque dispositivi basati sulla lettura di campi magnetici di bassissima intensità, per cui il rischio di malfunzionamento, per quanto ridotto, non è tra-

scurabile. Nel caso dei dischi una possibile contromisura contro il malfunzionamento è quello dell'uso del RAID (vedi sez. 6.1), ma questa copre soltanto un caso particolare di rischio, i guasti dell'hardware, e non le altre ragioni (come la cancellazione accidentale) per cui sono necessari i backup.

Al contrario dell'hardware l'affidabilità del software non sembra subire particolari miglioramenti, ed un programma solido, affidabile e sicuro è sempre un caso piuttosto raro. Inoltre l'affidabilità di un singolo programma è relativamente inutile se un altro va a sovrascrivere per errore i suoi dati. In questo la struttura di un sistema Unix, con la separazione dei privilegi, e l'uso di utenti distinti può essere di aiuto, ma di nuovo non è la soluzione al problema di eventuali perdite di dati.

L'affidabilità delle persone è sempre una variabile difficile da quantificare, sia in termini delle competenze che riducano l'eventualità di errori nelle operazioni, che però possono sempre accadere anche al più esperto degli amministratori, sia nella onestà e correttezza della singola persona rispetto al caso di eventuali azioni dolose che possono causare, direttamente o indirettamente, la distruzione dei dati o il danneggiamento dei relativi supporti.

Per quanto riguarda i disastri naturali, ed altre eventualità remote completamente distruttive di ogni tipo, occorre sempre valutare quanto la distruzione completa dei propri dati all'interno di un evento catastrofico possa incidere in termini di perdite complessive e determinare se vale la pena di prevedere delle contromisure specifiche per tali evenienze, che spesso comportano un forte aggravio di costi sia in termini di attrezzature che di gestione.

Come accennato, mantenere dei backup è la strategia più semplice ed efficace per proteggersi dalle molteplici evenienze che possono causare la perdita di dati. Avere a disposizione copie multiple dei propri dati rende meno critica la perdita di una di esse, ed il costo della perdita si riduce semplicemente a quello del ripristino degli stessi dal backup, e non alla ricreazione da zero degli stessi.

Perché questa strategia sia efficace occorre però che i backup siano effettuati in maniera appropriata; come ogni altra cosa del mondo reale infatti essi possono fallire ed al solito tenderanno a farlo nel peggior modo e momento possibili, per cui potreste trovarvi senza disco e scoprire che un nastro è illeggibile, o che vi siete dimenticati di inserire nel backup alcune informazioni essenziali, o che l'unica unità a nastro in grado di rileggere i vostri backup si è rotta pure lei.

Per questo motivo nell'effettuare dei backup il primo punto da prendere in considerazione è quello di pianificare una opportuna strategia. Ci sono infatti parecchie variabili da tenere sotto controllo, come la quantità di dati che si vogliono mantenere, la frequenza dei backup, il periodo che si intende coprire con il backup, il mezzo utilizzato, ed infine le modalità di gestione dei backup stessi.

In genere un primo passo è quello di definire cosa si intende inserire nel backup: ad esempio non è quasi mai strettamente indispensabile effettuare un backup dei programmi di sistema, se non nel caso in cui una reinstallazione da zero non comporti un carico di lavoro eccessivo, mentre può essere importante salvare delle configurazioni ottenute come risultato finale di parecchie ore di lavoro. Sono senz'altro da salvare tutti i dati di sistema, gli archivi della posta e di un eventuale database e del web, e quant'altro non è replicabile in maniera automatica, insieme ai dati personali dei singoli utenti. Questo ad esempio vuol dire, nell'ambito di un sistema GNU/Linux, che si devono salvare i contenuti delle directory `/etc`, `/var` e `/home`, mentre non è necessario salvare quelli di `/usr`.

Una volta determinata la dimensione totale dei dati che si vogliono archiviare, occorre stabilire anche il periodo di tempo che si intende coprire con i backup, e la frequenza con cui si effettuano questi ultimi. Se infatti si avesse un sistema di backup contenente solo le informazioni necessarie a ripristinare la situazione ad un certo istante, non si potrebbero recuperare eventuali dati cancellati in precedenza. Nella pratica invece il caso più comune di recupero di dati da un backup è proprio quello di dati cancellati inavvertitamente, che, se assenti alla data dell'ultimo backup disponibile, sarebbero irrimediabilmente perduti.

Questo comporta che deve essere determinata la frequenza con cui si vogliono effettuare i backup, in modo da poter essere in grado di ricostruire la situazione dei propri dati ad istanti diversi nel passato. La frequenza dei backup dipende ovviamente anche da quella con cui variano i dati: in genere le configurazioni di sistema o un indirizzario variano senz'altro molto meno della posta elettronica o dei dati di accesso a servizi web. Un altro fattore che incide sulla frequenza è il tempo che occorre a creare il backup, dato che se questo dovesse eccedere il periodo fra due backup successivi si avrebbe una situazione in cui un backup non si completa prima dell'inizio del successivo, e non avrebbe senso usare una frequenza maggiore.

Infine la scelta del periodo da coprire con i backup dipende dall'uso degli stessi, ad esempio se servono solo a mantenere una copia di riserva che permetta il recupero di eventuali cancellazioni accidentali passate inosservate, o se si vuole mantenere una storia completa per ragioni di archiviazione. Ovviamente più lungo sarà il periodo, più risorse dovranno essere utilizzate nella manutenzione dei backup. Inoltre anche qualora non esistano esigenze di archiviazione, occorre tenere presente che se una cancellazione accidentale è rimasta inosservata per un periodo di tempo superiore a quello di cui si mantiene il backup, questa non sarà più recuperabile.

Un'altra decisione cruciale nella strategia di backup è la scelta del supporto da utilizzare per gli stessi. Le variabili in gioco sono il costo, l'affidabilità, la velocità e l'utilizzabilità. Il costo incide direttamente sulla quantità di dati che si possono salvare, per cui è opportuno avere un mezzo poco costoso. L'affidabilità invece è fondamentale, dato che un backup inaffidabile è assolutamente inutile, ma l'affidabilità dipende anche dal tipo di problema che ci si trova ad affrontare: ad esempio i dischi sono affidabili, ma a lungo non sono stati considerati adatti come mezzi di backup in quanto vincolati ad essere residenti nello stesso computer che contiene i dati originali, che può andare distrutto, danneggiato o manomesso.

La velocità è senz'altro utile, ma diventa cruciale solo qualora un mezzo troppo lento rendesse impossibile effettuare i backup con la frequenza necessaria. L'utilizzabilità viene spesso data per scontata, ma occorre invece riflettere su quanto possa essere pericoloso avere uno splendido sistema di backup perfettamente funzionante, che funziona però solo con una vecchia scheda hardware fuori produzione.

Tenendo conto di tutte queste caratteristiche a lungo gli unici supporti considerati appropriati per i backup erano i nastri magnetici. I CD o i DVD masterizzati infatti, benché funzionali e relativamente poco costosi, hanno grossi problemi di stabilità nel lungo periodo, i supporti si deteriorano piuttosto rapidamente, e non danno garanzie già dopo pochi anni, per questo sono inadatti per l'archiviazione.

La soluzione classica era allora quella di utilizzare unità a nastro SCSI (vedi sez. 5.4.2). I nastri infatti hanno una ottima stabilità nel tempo se si usano supporti di qualità come i DLT ed i sistemi a nastro professionali, non altrettanto si può dire per i DAT, che hanno grossi problemi di stabilità. I nastri inoltre consentono di immagazzinare grandi quantità di dati in relazione alle dimensioni medie dei dischi, anche se oggi, con l'espandersi della capacità di questi ultimi,

questo è sempre meno vero. Infine le interfacce SCSI hanno dimostrato di mantenersi utilizzabili nell'evoluzione della tecnologia, il principale difetto resta il costo piuttosto elevato.

Oggi, con la disponibilità a basso costo di dischi rimuovibili su bus USB, le cose stanno cambiando. Questi infatti si stanno affermando sempre di più come alternativa ai nastri, rispetto ai quali presentano l'innegabile vantaggio di un costo iniziale nettamente inferiore, non essendo necessario per il loro funzionamento alcun dispositivo aggiuntivo. Lo svantaggio può esserci in caso di necessità di archiviazione di grandi quantità di dati, per i quali i nastri presentano una maggiore robustezza meccanica, e, su numeri elevati dei supporti da mantenere, un costo inferiore.

Oltre alla scelta dei supporti una strategia di backup deve comunque anche prevedere la definizione delle modalità di conservazione degli stessi. Ad esempio può non essere troppo saggio tenere i propri supporti in un armadio nel corridoio, sotto una vecchia tubatura del riscaldamento che potrebbe rompersi. Inoltre qualunque sia la modalità di stoccaggio utilizzata occorre anche prevedere verifiche periodiche del funzionamento dei backup (effettuando dei ripristini di controllo), dato che accorgersi che qualcosa è andato storto nel momento del bisogno non è molto simpatico.

A tal proposito viene citato l'esempio di un sistemista di un paese del nord Europa che aveva accuratamente pianificato tutta la sua strategia, prevedendo anche lo stoccaggio dei nastri in un locale sicuro in una locazione remota. Questi si accorse la prima volta che dovette recuperare dei dati che tutti i suoi nastri erano stati rovinati dal campo magnetico generato dai fili del riscaldamento del sedile della macchina su cui appoggiava i nastri quando li portava al sito di stoccaggio.

Una volta esaminati i principali rischi da cui ci si deve difendere, determinati quali dati si intende proteggere e prese in considerazione le caratteristiche delle possibili risorse per eseguire questa operazione si potrà iniziare a pianificare una strategia di backup. In genere si classificano i backup in tre diverse categorie generali:

***backup completo* (*full backup*)**
: si chiama così la realizzazione di una copia completa di tutti i dati che interessano, eseguita ad un certo istante. Il vantaggio di disporre di un backup completo è che questo è sufficiente da solo a garantire il ripristino dei dati al momento in cui è stato eseguito. Lo svantaggio è che in genere l'esecuzione di una copia completa di tutti i dati richiede un tempo non trascurabile; inoltre per poter garantire la copertura con il backup per un certo periodo di tempo occorre duplicare una grande quantità di dati che per lo più saranno identici, utilizzando in maniera sostanzialmente inefficiente le risorse su cui si salvano i dati. Esistono però soluzioni specifiche che consentono di mantenere comunque una copia unica degli stessi file, posto che questi siano mantenuti su dei dischi, se si usano i nastri questo tipo di strategia non può essere invece realizzata.

***backup incrementale* (*incremental backup*)**
: si chiama così la realizzazione di una copia di tutti i dati che sono stati modificati a partire dall'ultimo backup eseguito (completo o incrementale che sia). In questo caso cioè non si ha una copia completa dei dati, ma solo di quello che è cambiato rispetto all'ultima esecuzione, per cui per poter eseguire un ripristino sarà necessario non soltanto avere l'ultimo backup incrementale eseguito, ma anche tutti quelli precedenti

fino all'ultimo backup completo disponibile. Il vantaggio dei backup incrementali è che permettono di limitare l'occupazione di risorse ed i tempi di esecuzione dei backup, mantenendo la capacità di ricostruire la situazione ad un dato istante per un certo periodo di tempo senza dover registrare tutte le volte l'insieme completo dei dati. Lo svantaggio è che si complica la procedura di recupero dei dati, che necessiterà di un numero di passaggi crescente con il numero di backup incrementali coinvolti, e si diminuisce l'affidabilità perché il fallimento di un backup incrementale renderà inutilizzabili tutti quelli successivi basati su di esso.

***backup differenziale* (*differential backup*)**
si chiama così la realizzazione di una copia di tutti i dati che sono stati modificati a partire dall'ultimo *backup completo*. Un backup differenziale è un compromesso fra un backup completo ed un backup incrementale: non richiede di avere tutti i backup parziali precedenti dall'ultimo backup completo, ma solo quest'ultimo. Pertanto si riduce l'occupazione delle risorse ed il tempo impiegato rispetto ad un backup completo, ma in maniera inferiore rispetto a quanto si sarebbe ottenuto con un backup incrementale. Rispetto a quest'ultimo però, non essendo necessari i backup differenziali precedenti, ma solo l'ultimo backup completo, si ha il vantaggio di una maggior sicurezza e di una procedura di ripristino più semplice.

In generale una strategia di backup si basa sulla opportuna combinazione di backup completi e backup incrementali o differenziali. Si stabilisce una opportuna cadenza a cui eseguire dei backup completi, e si copre il periodo di tempo dall'uno all'altro con backup incrementali (o differenziali). In un mondo ideale, con risorse infinite, la soluzione è eseguire una sequenza continua di backup completi mantenuti indefinitamente, così da avere il massimo della sicurezza.

Nel mondo reale però ci si scontra con alcuni limiti, il primo dei quali è il tempo necessario alla esecuzione del backup: questo già pone un limite insormontabile alla frequenza dei backup completi. Si tenga presente inoltre che in molti casi l'esecuzione del backup richiede l'interruzione del servizio di cui si stanno salvando i dati e non è detto che si possa restare in questo stato troppo a lungo. Questo avviene perché, ad esempio con i database, se si copiano i dati a servizio attivo questi potrebbero essere in uno stato inconsistente, a causa dell'accesso concorrente fatto dal programma che esegue il backup, che li legge, e di quello che eroga il servizio, che li modifica.

Questo è il motivo per cui in genere le procedure di backup vengono eseguite a notte fonda, onde diminuire al massimo gli eventuali disagi causati dal fermo dei servizi. Per risolvere questo problema può essere di grande aiuto l'uso dei cosiddetti *snapshot* del filesystem, in cui si genera una sorta di *fotografia* ad un certo istante dello stato dei propri file, accessibile in maniera opportuna anche successivamente (ne vedremo un esempio in sez. 6.2). In questo caso si potrà fermare il servizio solo per il tempo necessario a creare lo *snapshot* per poi eseguire il backup su di esso in tutta calma.

Il secondo limite è quello dello spazio disponibile per l'archiviazione dei dati del backup, che non è mai infinito, ed in genere tende a finire assai prima di quanto si era previsto inizialmente. Per questo motivo si ricorre quasi sempre a backup incrementali, che oltre a consentire una esecuzione più veloce limitano la quantità di spazio occupato.

A titolo di esempio una strategia di backup comunemente utilizzata è quella di effettuare backup incrementali a frequenza giornaliera, intervallati da backup completi a cadenza settimanale, mantenendo uno storico mensile, ed eventualmente archiviando i backup mensili per un

periodo di tempo più o meno lungo a secondo delle esigenze di "*data retention*" che si possono avere.

Si ricordi infine, ed è opportuno sottolinearlo una seconda volta dato che questo aspetto viene spesso sottovalutato, che una strategia di backup non può definirsi tale se non prevede anche in maniera esplicita, oltre alla esecuzione dei backup, una procedura per la verifica periodica del loro ripristino, che assicuri l'effettiva disponibilità dei dati in essi archiviati quando questi dovranno essere recuperati.

4.1.2 I comandi `tar` e `cpio`

Il comando più usato per la gestione di archivi di file è `tar`, chiamato così dall'inglese "***t**ape **ar**chive*" dato che il comando è nato proprio per archiviare i file su nastro magnetico. Al di là del suo uso per la creazione di archivi, `tar` è anche il programma più diffuso per la distribuzione di pacchetti di software dai sorgenti (come vedremo in sez. 4.2.1). La caratteristica principale del comando è che è in grado di salvare su un singolo file di archivio e ripristinare da esso il contenuto e la struttura di una o più directory, mantenendo invariate anche le caratteristiche specifiche di tutti file (come permessi, proprietari, tempi, ecc.) in esse presenti.

Il comando `tar` supporta una grande varietà di opzioni ed argomenti, e la versione GNU è stata dotata di una serie di ulteriori estensioni. In generale però si possono distinguere le opzioni del comando in due classi principali: quelle che indicano delle *operazioni* e quelle che specificano dei *parametri*. In ogni invocazione si può specificare non più di una operazione, mentre si possono avere un numero qualunque di parametri.

Opzione		Significato
`-c`	`--create`	crea un nuovo archivio.
`-x`	`--extract`	estrae i file da un archivio.
`-t`	`--list`	elenca i file contenuti in un archivio.
`-A`	`--concatenate`	aggiunge un archivio in coda ad un altro.
`-d`	`--diff`	confronta i membri di un archivio con la controparte sul filesystem e riporta ogni differenza in dimensioni, permessi, proprietari e tempi.
`-u`	`--update`	aggiunge file in coda all'archivio, ma solo se sono più recenti della versione già presente nell'archivio o non sono presenti in esso.
`-r`	`--append`	aggiunge file in coda all'archivio.
	`--delete`	cancella i file da un archivio (questo non può funzionare sui nastri).
	`--compare`	identica a `--diff`.
	`--get`	identica a `--extract`.

Tabella 4.1: Operazioni del comando `tar`.

Il comando prevede tre operazioni principali, `-c` per creare un archivio, `-t` per verificarne il contenuto, e `-x` per estrarlo, ma le operazioni sono in totale otto, e sono illustrate in tab. 4.1, sia in forma breve che estesa. Si è usata la sintassi delle opzioni nella forma prevista dal progetto GNU, ma `tar` è uno dei comandi storici di Unix, e ne esistono molte versioni. Per questo anche la versione GNU, la sola che tratteremo, supporta varie sintassi, ed in particolare si può sempre fare a meno di utilizzare il "`-`" quando si specifica una opzione in forma breve.

Come accennato oltre alle operazioni `tar` supporta una enorme quantità di opzioni che modificano il comportamento del programma. Di queste ne esistono due che sono di uso comune e vengono utilizzate quasi sempre; la prima è `-v` (o `--verbose`), che aumenta la *prolissità* del comando, facendogli stampare a video durante l'esecuzione molte più informazioni, ad esempio la lista dei file inseriti o estratti dall'archivio. É in genere possibile aumentare ulteriormente la quantità di informazioni stampate inserendo la stessa opzione una seconda volta.

La seconda è `-f` (o `--file`) che serve a specificare il nome del file da usare come archivio da cui leggere o su cui scrivere. Il comando infatti, se non la si utilizza, usa come default rispettivamente lo *standard input* e lo *standard output*, a meno che non sia stata definita la variabile di ambiente `TAPE`, nel qual caso viene usato il dispositivo indicato in detta variabile, che deve fare riferimento ad una unità a nastro.[1] Per evitare confusione è pertanto opportuno specificare sempre esplicitamente il nome dell'archivio con questa opzione.

Opzione	Significato
`-v`	aumenta la *prolissità* del comando facendogli stampare a video più informazioni riguardo le operazioni correnti.
`-f` *`file`*	indica il file di archivio a cui fare riferimento con l'operazione corrente.
`-z`	comprime al volo l'archivio utilizzando il programma `gzip`.
`-j`	comprime al volo l'archivio utilizzando il programma `bzip2`.
`-J`	comprime al volo l'archivio utilizzando il programma `xz`.
`-O`	estrae i file sullo *standard output*.
`-M`	opera su un archivio multivolume.
`-G`	esegue il comando in modalità incrementale (vecchio formato).
`-g` *`list`*	esegue il comando in modalità incrementale nel nuovo formato, appoggiandosi alla lista *`list`*.
`-C` *`dir`*	esegue le operazioni sulla directory *`dir`* invece che su quella corrente.

Tabella 4.2: Principali opzioni generiche del comando `tar`.

Un'altra delle funzionalità offerte da `tar` è quella che consente, appoggiandosi ai comandi trattati in sez. 2.2.4, di eseguire in maniera automatica la compressione (in creazione) e la decompressione (in estrazione) degli archivi. A tale scopo l'opzione usata più comunemente è `-z` che effettua una compressione con `gzip`, ed è invalso l'uso comune di utilizzare una ulteriore estensione `.gz` al nome dell'archivio (che diventa qualcosa del tipo `archivio.tar.gz`) per identificare i file, detti in gergo "*tarball*", creati con questa opzione. Alternativamente si può utilizzare l'opzione `-j`, che consente di utilizzare il programma `bzip2` al posto di `gzip`, e in questo caso per gli archivi si usa l'ulteriore estensione `.bz2` ed analogamente con `-J` viene usato `xz` e gli archivi usano l'estensione `.xz`. Si può infine specificare un programma di compressione qualunque con l'opzione estesa `--use-compress-program` seguita dal nome del programma da usare.

Come accennato per creare un archivio il comando necessita dell'opzione `--create` o `-c`; in questo caso dovrà anche essere specificato con `-f` il nome del file su cui saranno archiviati i file, e se non si comprime di norma si utilizza l'estensione `.tar` per identificare questi file. L'opzione prende anche il valore "`-`" per indicare rispettivamente *standard input* e *stardard output*. Date queste due opzioni il comando prende come argomenti una lista dei singoli file da archiviare, se

[1] come vedremo in sez. 5.4.2 in genere i nastri vengono utilizzati tramite l'interfaccia SCSI, in tal caso valori possibili per `TAPE` sono `/dev/st0` o `/dev/nst0` (posto di avere un solo nastro) a seconda che si voglia il riavvolgimento automatico o meno ogni volta che il file viene chiuso.

però fra questi si inserisce una directory il comando archivierà anche il contenuto di quest'ultima, e di tutte le directory sottostanti. Ad esempio si possono archiviare alcuni file con:

```
piccardi@anarres:~/truedoc/corso$ tar -cvf dispense.tar *.tex
advadmin.tex
appendici.tex
...
stradmin.tex
struttura.tex
```

e si noti come si sia usata anche l'opzione `-v` per stampare la lista dei file archiviati; allo stesso modo si potrà archiviare tutta una directory con:

```
piccardi@anarres:~/truedoc$ tar -cvf dispense.tar corso
corso/
corso/struttura.aux
corso/esercizi.txt
corso/struttura.tex
corso/shell.aux
corso/images/
corso/images/menuconfig1.png
corso/images/lvm_struct.eps
...
corso/corso.tex
corso/ringraziamenti.tex
```

e si noti come in questo caso vengano archiviate ricorsivamente anche tutte le directory sottostanti ed il relativo contenuto.

Si tenga presente che quando si crea un archivio con `-c` il file di destinazione viene sovrascritto, se si vuole invece aggiungere dei file ad un archivio preesistente occorre usare l'opzione `-r`. Inoltre, a meno di non specificare un pathname assoluto, il file viene creato nella directory corrente, questo significa anche che se quest'ultima è inclusa nell'archivio `tar` si troverà ad affrontare il problema di come archiviare l'archivio stesso. Il comando è sufficientemente scaltro da accorgersi di questa situazione ed evitare di inserire nell'archivio che sta creando il file dell'archivio stesso:

```
piccardi@anarres:~/truedoc$ tar -cf dispense.tar .
tar: ./dispense.tar: file is the archive; not dumped
```

Una ultima caratteristica di `tar` è che qualora si utilizzino dei pathname assoluti per indicare le directory o i file da archiviare, questi vengono sempre salvati con un pathname relativo in cui la "`/`" viene eliminata, e si riceverà un avviso al proposito. Questo avviene perché quando si vanno ad estrarre i dati da un archivio tutti i file in esso contenuti vengono sempre ripristinati a partire dalla directory corrente, ed anche se se ne può specificare una diversa con l'opzione `-C`, l'estrazione considera in ogni caso i pathname dei file contenuti nell'archivio come relativi rispetto alla directory in cui essi verranno estratti per evitare sovrascritture dei file del sistema.

Una volta creato un archivio se ne potrà verificare il contenuto con `-t` o `--list` e di nuovo si dovrà utilizzare `-f` per indicare il file contenente l'archivio, che in questo caso sarà letto. Se non si passano altri argomenti il risultato sarà la stampa a video della lista di tutti i file contenuti nell'archivio, anche senza la presenza dell'opzione `-v`, specificando la quale si otterrà invece una lista in formato esteso, analogo a quella ottenibile con `ls -l`. Si può verificare la presenza di un singolo file o di una lista passando come argomento il nome (o i nomi), si tenga presente però che

in questo caso deve essere specificato il nome completo del file così come è stato archiviato (quello stampato dall'opzione `-v` in fase di creazione), il confronto infatti viene eseguito solo nei termini della corrispondenza fra l'argomento ed il nome come questo viene visto nell'archiviazione.

Per estrarre i dati inseriti in un archivio si deve utilizzare l'opzione `-x` o `--extract`, di nuovo è necessario specificare l'archivio da cui si estraggono i file con `-f`. Come per la creazione l'uso dell'opzione `-v` permette di ottenere la lista dei file che vengono estratti dall'archivio. L'estrazione avviene sempre nella directory corrente, e qualora si siano archiviate intere directory ne viene ricreata l'intera gerarchia. Se non si specifica nessun argomento il comando estrae il contenuto completo dell'archivio, ma è possibile estrarre un singolo file fornendone il nome (o il pathname se questo è collocato in una sottodirectory), che si può ottenere dalla lista fornita dall'opzione `-t`.

Si tenga presente infine che nell'estrazione il comando ricrea i file con tutte le caratteristiche che essi avevano nel sistema su cui sono stati creati; questo significa anche che vengono mantenuti i permessi e l'utente ed il gruppo proprietario del file, utente e gruppo che nel sistema corrente possono non esistere o essere del tutto diversi. Utente e gruppo però vengono mantenuti soltanto se l'archivio viene estratto dall'amministratore, gli utenti normali infatti (vedi sez. 1.4.3) non possono creare file con proprietario diverso da se stessi e con gruppo diverso da quelli a cui appartengono.

Di questo può essere necessario dover tenere conto quando si estraggono degli archivi creati da altri (come quelli dei pacchetti sorgenti di cui parleremo in sez. 4.2). Inoltre dato che alcuni filesystem (come il `vfat` di Windows) non supportano la presenza di permessi, utenti e gruppi, il programma può lamentarsi dell'impossibilità di ricreare questi attributi dei file. Un altro errore che viene segnalato, in caso di impostazione sbagliata dell'ora di sistema, è quando i file che vengono creati hanno dei tempi situati nel futuro.

Una delle capacità più interessanti di `tar` è quella di saper gestire backup incrementali con l'uso dell'opzione `-g`. In questo caso l'opzione prende come parametro il nome di un file, detto di *snapshot*, su cui `tar` si appoggia per determinare quali sono stati i file modificati dall'ultimo backup. Per questo vengono utilizzate le date di ultima modifica per cui è essenziale che il tempo di sistema (vedi sez. 2.4.3) non sia stato modificato inopportunamente.

Alla prima esecuzione, quando il file non esiste, questo viene creato ed alle successive esecuzioni sarà possibile creare nuovi archivi contenenti solo i file modificati dopo l'esecuzione del backup a cui fa riferimento il file di *snapshot*. Ad ogni esecuzione con `-g` il file di *snapshot* viene aggiornato con tutti i cambiamenti intercorsi, per cui sarà possibile, utilizzando sempre lo stesso file, eseguire backup incrementali. È inoltre possibile, utilizzando una copia del file di *snapshot* creato la prima volta nell'esecuzione del backup completo, effettuare backup differenziali.

Un secondo comando usato storicamente nei sistemi unix-like per la gestione dell'archiviazione di file e directory è `cpio`, che come `tar` permette di estrarre ed inserire file e directory all'interno di un singolo file di archivio, per il quale è anche in grado di usare lo stesso formato usato da `tar`. A differenza di `tar` però `cpio` prende sempre come argomento di ingresso una lista di file.

Il comando `cpio` ha tre modalità operative, che devono essere sempre indicate ad ogni invocazione del programma. In modalità *copy-out* il comando copia i file dentro l'archivio e per attivarla deve essere specificata l'opzione `-o` (o `--create`). Il programma legge una lista di file dallo *standard input* e scrive l'archivio risultante sullo *standard output*, ma è uso comune ge-

nerare la lista direttamente con l'uso di `find` o `ls`. Si potranno cioè replicare le operazioni di archiviazione viste in precedenza con `tar` con qualcosa del tipo:

```
piccardi@anarres:~/truedoc/corso$ ls *.tex | cpio -ov > dispense.cpio
advadmin.tex
appendici.tex
...
stradmin.tex
struttura.tex
3974 blocks
```

mentre per archiviare tutta una directory si dovrà eseguire:

```
piccardi@anarres:~/truedoc$ find corso -depth -print | cpio -ov > dispense.cpio
corso/struttura.aux
corso/esercizi.txt
corso/struttura.tex
corso/shell.aux
corso/images/menuconfig1.png
corso/images/lvm_struct.eps
...
corso/config.aux
corso/corso.tex
corso/ringraziamenti.tex
corso
80540 blocks
```

In modalità *copy-in* il comando estrae i file contenuti in un archivio o ne elenca il contenuto. Per attivare questa modalità deve essere specificata l'opzione `-i` o `--extract`. L'archivio viene letto dallo *standard input*, se non si specifica nessuna opzione tutti i file vengono estratti, altrimenti tutti gli argomenti che non sono opzioni vengono considerati come *pattern* secondo la sintassi[2] del *filename globbing* della shell (vedi sez. 2.1.3) e vengono estratti solo i file che corrispondono. Si tenga presente che di default il comando non crea le directory, per cui se si vuole estrarre il contenuto di un archivio che contiene un intero albero di directory occorre utilizzare l'opzione `-d`. Una invocazione tipica in questa modalità è pertanto qualcosa del tipo:

```
cpio -idv < dispense.cpio
```

In modalità *copy-pass* il comando copia dei file da una directory ad un altra, combinando in sostanza le due modalità *copy-in* e *copy-out* in un unico passo senza creare un archivio; per attivare questa modalità deve essere specificata l'opzione `-p` o `--pass-through`. Il comando legge la lista dei file da copiare dallo standard input e li copia nella directory che deve essere specificata come argomento non opzionale. Pertanto una invocazione tipica in questa modalità è del tipo:

```
cpio -pdv < lista  directory
```

Il comando, a parte quelle usate per specificarne la modalità di operazione, supporta molte altre opzioni. In particolare con `-v` si richiede una maggiore prolissità nelle operazioni, con la stampa dei nomi dei file che vengono processati; con `-t` si richiede di stampare la lista del contenuto di un archivio, con `-A` di aggiungere i file ad un archivio esistente (invece di sovrascriverlo).

[2] con due eccezioni, vengono considerati validi come caratteri sui quali verificare una corrispondenza anche un "`.`" all'inizio del file e il carattere "`/`", così da considerare nella corrispondenza anche i file nascosti e intere sezioni di pathname.

Opzione	Significato
`-0`	prende in ingresso una lista di stringhe terminate dal carattere NUL (generabili con `find -print0`) per permettere l'archiviazione di file il cui nome contiene caratteri speciali.
`-a`	ripristina il tempo di ultimo accesso sui file appena letti perché non risultino tali.
`-d`	ricrea le directory in cui si trovano i file.
`-f`	copia solo i file che non corrispondono al pattern specificato.
`-F`	permette di specificare un nome di file da usare come archivio al posto dello *standard input* o dello *standard output*.
`-m`	preserva i precedenti tempi di ultima modifica quando ricrea i file in fase di estrazione.
`-t`	stampa la lista dei file contenuti nell'archivio.
`-u`	sovrascrive eventuali file già esistenti con quelli dell'archivio senza chiedere conferma.
`-v`	stampa informazioni aggiuntive sui file che vengono processati dal comando (in combinazione con `-t` viene usato un formato analogo a quello di `ls -l`).

Tabella 4.3: Principali opzioni del comando `cpio`.

L'elenco delle principali opzioni di `cpio` è riportato in tab. 4.3, in cui si sono tralasciate le tre opzioni principali per selezionare le modalità di operazione; al solito l'elenco completo delle opzioni è disponibile nella pagina di manuale accessibile con `man cpio`, anche se la documentazione è mantenuta principalmente nella pagina info accessibile con `info cpio`.

4.1.3 I comandi `dump` e `restore`

I comandi `tar` e `cpio` illustrati in precedenza vengono usati per archiviare liste di file o directory sulla base di quanto specificato al comando, per i backup esistono però anche dei comandi, come `dump` e `restore` che permettono di effettuare l'archiviazione a livello di un intero filesystem.

Il limite di un programma come `dump` è che questo funziona solo per un filesystem che ne supporti le operazioni salvando le informazioni necessarie. Questo nel caso di Linux è vero solo per il filesystem *ext2* e la sua estensione *journalled*, *ext3*. Inoltre per poter utilizzare questa funzionalità occorre anche montare opportunamente il filesystem, e come vedremo in sez. 5.1.3, per questo il file `/etc/fstab` prevede un campo apposito, il quinto, che deve essere inizializzato opportunamente ad un valore non nullo.

Una volta abilitato il supporto per il *dump* del filesystem tramite `fstab` effettuando operazioni di backup dello stesso tramite `dump` saranno salvate le informazioni necessarie a ricostruire quali file sono stati cambiati da un backup all'altro, così da permettere dei backup incrementali. Il funzionamento del programma prevede la possibilità 10 diversi livelli di *dump*, numerati da 0 a 9, da specificare con le opzioni omonime (cioè `-0`, `-1`, ... `-9`), e se non viene specificato nulla viene usato il livello 9.

Il livello 0 indica un backup completo, in cui si sono salvati tutti i file, e stabilisce un punto di partenza per tutti gli altri livelli. Un livello superiore richiede il salvataggio di tutti i file che sono cambiati dall'ultimo backup effettuato con un livello inferiore. Questo significa che un backup di livello 9 salverà sempre solo i file modificati dopo l'ultimo backup precedente.

Quindi per fare un esempio,[3] se si esegue un backup con livello 2 seguito da uno di livello

[3]si suppone che prima si sia partiti con un dump di livello 0, un passaggio iniziale infatti è sempre necessario.

4 si avranno in quest'ultimo solo i file cambiati dal momento in cui si è eseguito il precedente backup di livello 2; se poi si eseguisse un dump di livello 3 sarebbero salvati di nuovo tutti i file modificati dopo il backup di livello 2 (e il livello 4 sarebbe ignorato), se al posto del dump di livello 3 se ne fosse fatto uno di livello 5 invece si sarebbero salvati solo i file modificati dopo il backup di livello 4.

Questo permette di poter eseguire i backup incrementali in maniera efficiente per ridurre al minimo sia il tempo di recupero che il numero di supporti utilizzati. Una possibile strategia di backup è quella riportata nella pagina di manuale, in cui si fanno backup incrementali giornalieri, usando un insieme di supporti da ruotare settimanalmente, usando un algoritmo di tipo *torre di Hanoi* modificato.

Opzione	**Significato**
`-D`	permette di cambiare il file su cui viene mantenuta l'informazione relativa ai precedenti backup (completi o incrementali) che di solito è `/var/lib/dumpdates`.
`-e`	esclude dal backup una lista di *inode* (vedi sez. 1.2.1) passati per numero.
`-E`	esclude una lista di *inode* elencati nel file specificato come parametro.
`-f`	scrive il backup sul file passato come parametro, si possono specificare più file separandoli con virgole, e saranno considerati come volumi successivi.
`-L`	usa la stringa passata come parametro come etichetta del backup.
`-M`	abilita il salvataggio multivolume (in cui si usa `-f` per specificare più di un volume di backup).
`-u`	aggiorna il file `/var/lib/dumpdates` che contiene la tabella in cui sono scritti i filesystem per i quali cui sono stati effettuati i backup, e le rispettive date ed il livello.
`-v`	aumenta la *prolissità* del comando facendogli stampare a video più informazioni riguardo le operazioni correnti.

Tabella 4.4: Principali opzioni del comando `dump`.

A parte l'opzione che ne specifica il livello il comando richiede la presenza dell'opzione `-f` seguita da un parametro che indica il file su cui effettuare il backup (in genere si usano le unità a nastri, ma si può specificare un file ordinario o usare `-` per indicare lo *standard output*), e prende come argomento il *mount point* del filesystem di cui si vuole eseguire il backup. In realtà si può anche specificare una directory contenente i file che si vogliono archiviare e non un *mount point*, in questo caso però sono consentiti solo backup completi di livello 0 e tutto il contenuto della directory deve risiedere nello stesso filesystem. Oltre quelle citate `dump` prende numerose altre opzioni, molte delle quali sono relative alla gestione dei nastri su cui usualmente vengono salvati i backup. Si sono riportate le principali in tab. 4.4, per l'elenco completo si può al solito fare riferimento alla pagina di manuale.

Il comando che permette di recuperare i dati archiviati con `dump` è `restore`, che esegue esattamente il compito inverso, e come `dump` usa l'opzione `-f` per indicare il file (ordinario o di dispositivo) da cui recuperare i dati da ripristinare. Il comando inoltre prevede la necessaria presenza di una serie di opzioni che ne indicano la modalità di operazione. Le principali sono `-C` che effettua un confronto fra i file passati come argomento e quelli sul backup, `-i` che porta

in modalità interattiva (dove possono essere dati una serie di comandi, descritti nella pagina di manuale), `-r` che permette di ricostruire un intero filesystem da zero, `-t` che verifica se i file passati come argomento sono nel backup o stampa a video il contenuto integrale dello stesso se invocato senza argomenti, e `-x` che effettua il ripristino dal contenuto dell'archivio: di una directory, se questa viene passata come argomento, o di tutto l'archivio se non vengono passati argomenti.

Opzione	**Significato**
`-f`	indica il file in cui è stato memorizzato il backup.
`-C`	esegue un confronto fra i file presenti nel filesystem e quelli nel backup.
`-i`	attiva la modalità interattiva.
`-r`	ricostruisce un intero filesystem da zero.
`-t`	presenta un elenco di file nel backup.
`-x`	estrae una directory dal backup (o tutto il contenuto).
`-h`	estrae il contenuto di una directory, e non di quelle in essa contenute.
`-M`	abilita il ripristino da un archivio multivolume.
`-v`	stampa una maggiore quantità di informazione durante le operazioni.

Tabella 4.5: Principali opzioni del comando `restore`.

Come `dump` anche `restore` prevede numerose opzioni, parecchie delle quali servono a controllare proprietà relative all'uso dei nastri. Oltre a quelle brevemente spiegate in precedenza si sono riportate in tab. 4.5 le più rilevanti; al solito tutti i dettagli e l'elenco completo delle opzioni (e dei comandi disponibili in modalità interattiva) sono riportati nella pagina di manuale.

4.1.4 Altri comandi per il backup

Si sono raccolti in quest'ultima sezione alcuni comandi specialistici utilizzati per effettuare delle copie di dati che risultano di un certo interesse quando si devono eseguire dei backup; in questo caso si tratta sempre di backup completi, realizzati con la semplice esecuzione di una copia di sicurezza dei dati che interessano. Se infatti è sempre possibile usare il semplice `cp` per copiare dei dati (in genere con l'opzione `-a` se si vogliono preservare anche le proprietà dei file e copiare intere directory) esistono alcune esigenze specifiche che vengono coperte in maniera più efficace ricorrendo ad altri comandi.

Il primo di questi comandi è `dd` che viene normalmente utilizzato quando si devono effettuare delle copie del contenuto di interi dispositivi (dischi, floppy, chiavette USB ed anche CDROM). La principale differenza di `dd` rispetto a `cp` è che questo è nato, ed è presente fino dalle prime versioni di Unix, per effettuare delle copie a basso livello dei dati presenti sui dispositivi a blocchi, e consente di eseguire letture e scritture con buffer dati di dimensioni precise, in modo che queste corrispondano alle dimensioni dei blocchi dei dispositivi su cui va ad operare.

Dato che, come illustrato in sez. 1.2.1, nei sistemi unix-like "*tutto è un file*", per copiare i dati di un dispositivo si potrebbero comunque usare anche comandi ordinari come `cp`, `dd` fornisce però un grado di controllo molto maggiore. Inoltre il programma consente di specificare quanti dati copiare, sia in termini di numero di byte che di blocchi che di punto di partenza della copia,

ed è per questo molto utile quando si devono effettuare copie di parti specifiche di un disco, come il *Master Boot Record* (vedi sez. 5.1.2).

Date le sue origini agli albori di Unix il comando ha una sintassi abbastanza peculiare, che non usa le normali opzioni, e prevede invece che tutti gli argomenti siano passati in forma di assegnazione (con il carattere "`=`") di valori ad altrettanti operatori che ne controllano il comportamento. Se non si specificano argomenti il comando legge dallo *standard input* e scrive sullo *standard output*, ma usato così non è di grande utilità.

Per questo motivo il file da cui leggere e quello su cui scrivere vengono in genere indicati esplicitamente con gli operatori `if` ed `of`, per cui un'invocazione tipica del comando, con il quale è ad esempio possibile effettuare una copia fisica del contenuto di una chiavetta USB, sarà qualcosa del tipo:[4]

```
dd if=/dev/sdb of=usbkey.img
```

ed invertendo i valori dei due operatori si potrà effettuare il ripristino dalla copia. Si faccia comunque molta attenzione quando si va a scrivere su un dispositivo, perché il comando non chiede nessuna conferma, ed in caso di errore si rischia di distruggere totalmente il contenuto di un disco.

In questo modo il comando usa il valore di default di 512 byte per la dimensione dei blocchi; questa si può impostare con l'operatore `bs` a cui assegnare sia un semplice valore numerico (che specifica la dimensione in byte), che un valore con un prefisso che consente di indicare kilobyte, megabyte, ecc. Il comando supporta prefissi diversi per indicare sia i kilobyte metrici (1000 byte, con `Kb`) che gli usuali kilobyte "*informatici*" (1024 byte, con `K`); lo stesso vale per le altre dimensioni (con `M`, `G`, ecc.), per i dettagli si consulti la pagina di manuale.

Altri operatori importanti sono `count`, che consente di indicare quanti blocchi copiare, `skip` che consente di indicare un numero di blocchi dall'inizio del file da saltare nella lettura, e `seek` che esegue la stessa operazione sul file su cui scrivere. In questo modo è possibile fare eseguire a `dd` operazioni di copia di sezioni specifiche di un file su altre sezioni specifiche di un altro file. Si sono riportati i principali operatori in tab. 4.6, per l'elenco completo, che comprende anche quelli che consentono di effettuare al volo alcune conversioni di dati, si consulti al solito la pagina di manuale.

Operatore	**Significato**
`if`	file di ingresso da cui leggere i dati.
`of`	file di uscita su cui scrivere i dati.
`count`	numero di blocchi da leggere/scrivere.
`seek`	numero di blocchi da saltare nel file di uscita prima di iniziare la scrittura.
`skip`	numero di blocchi da saltare nel file di ingresso prima di iniziare la lettura.
`bs`	dimensione di un blocco di dati (512 byte di default).
`obs`	dimensione di un blocco di dati per la lettura.
`ibs`	dimensione di un blocco di dati per la scrittura.

Tabella 4.6: Principali operatori del comando `dd`.

[4] il file di dispositivo potrebbe anche essere un altro, a seconda della presenza o meno di altri dischi (vedi sez. 5.1.1), ovviamente si dovranno avere gli opportuni premessi di accesso al dispositivo, che normalmente sono disponibili solo all'amministratore.

Si tenga presente però che per effettuare correttamente la copia a basso livello del contenuto di un dispositivo a scopo di backup occorre che il filesystem presente nel dispositivo che si va a copiare non sia montato, in caso contrario infatti i dati saranno inconsistenti.[5] È per questo motivo che per effettuare un backup a livello di dispositivo normalmente si usano i comandi `dump` e `restore` di sez. 4.1.3, se però il dispositivo non è montato o è montato in sola lettura (vedi sez. 5.1.3) la coerenza è assicurata e si otterrà una copia corretta.

Un secondo comando molto utile per la realizzazione di backup è `rsync`. Il comando, come indica il nome, nasce per mantenere sincronizzato il contenuto di singoli file o intere directory sia in locale che in remoto. Rispetto all'uso di comandi come `tar` e `cp`, oltre alla capacità di lavorare via rete, ha il vantaggio di essere in grado di mantenere al minimo la quantità di dati trasferita. Il programma infatti funziona operando soltanto sui file che sono stati modificati, e trasmettendo, grazie un algoritmo sofisticato che gli consente di determinarle, soltanto le differenze presenti fra le due versioni, e non l'intero contenuto.

Per questo motivo `rsync` è molto utile quando si deve eseguire una copia di dati via rete o si ha a che fare con un mezzo lento come ad esempio un disco USB. Se infatti si dispone già di una versione precedente di quanto si vuole copiare, il comando trasferirà solo le differenze consentendo una operazione di copia molto più veloce.

La sintassi di base del comando è analoga a quella di `cp` e prevede come argomenti una o più sorgenti ed una destinazione; qualora ci siano più sorgenti (vale a dire più di due argomenti) la destinazione deve essere una directory. Il comando può essere usato anche con un solo argomento, nel qual caso fornisce la lista dei file presenti nella sorgente. Se usato localmente sorgenti e destinazione non sono altro che dei pathname, ma il comando consente di utilizzare una sorgente o una destinazione remota,[6] nel qual caso la forma generica dell'argomento sarà qualcosa del tipo:

```
user@machinename:/path/to/dest_or_src
```

la sintassi del comando però richiede che quando la sorgente è remota questa debba essere unica e la destinazione debba essere locale.

L'uso più comune di `rsync` è quello che prevede due directory come argomenti, per sincronizzare il contenuto della prima sulla seconda. Si tenga presente inoltre che quando si specifica come sorgente una directory il comando si comporta diversamente a seconda che questa sia indicata o meno con una "`/`" finale, nel primo caso infatti nella destinazione viene copiato solo il contenuto della directory, mentre nel secondo caso anche la directory stessa viene copiata all'interno della directory di destinazione. In entrambi i casi gli attributi della directory sorgente vengono comunque trasferiti sulla destinazione.

In generale quando lo si usa per eseguire copie il comando viene invocato con l'opzione `-a`, che consente di preservare link simbolici, file di dispositivo e tutti gli attributi (permessi, proprietari, tempi) di file e directory. Una seconda opzione comune, usata soprattutto per i trasferimenti via rete, è `-z` che attiva la compressione dei dati, mentre con `-v` si aumenta la prolissità del comando. Infine con `-u` si richiede al comando di non sincronizzare i file che risultano più recenti, come tempo di ultima modifica, nella destinazione.

[5] se il filesystem è montato il kernel lo aggiornerà in maniera autonoma, e non c'è nessuna garanzia sulla coerenza dei dati che potranno essere visti da `dd` nel momento in cui il programma li legge.

[6] non entriamo qui nei dettagli delle modalità del trasferimento via rete, se non per dire che normalmente questo avviene attraverso l'uso di SSH, che tratteremo in sez. 8.3.

Un elenco delle altre opzioni più significative è riportato in tab. 4.7, ed al solito per una documentazione più completa si può far riferimento alla pagina di manuale. Vale però la pena evidenziare due opzioni utilizzabili solo in forma estesa; con `--delete` si può richiedere la cancellazione nella destinazione dei file non presenti nella sorgente (il default è lasciarli) mentre con `--exclude` si può indicare un insieme di file da non trasferire, questa opzione prende come argomento una espressione analoga a quelle del *filename globbing* la cui sintassi è supportata integralmente, insieme ad alcune estensioni descritte in dettaglio nella pagina di manuale.

Opzione	**Significato**
`-a`	effettua un trasferimento in modalità archivio, equivalente a `-rlptgoD`.
`-r`	effettua una copia ricorsiva del contenuto della destinazione sulla sorgente.
`-l`	copia i link simbolici come tali.
`-p`	preserva i permessi.
`-t`	preserva i tempi.
`-g`	preserva il gruppo proprietario.
`-o`	preserva l'utente proprietario.
`-D`	preserva file speciali e di dispositivo.
`-u`	non esegue la sincronizzazione se a destinazione esiste una versione più recente.
`-v`	aumenta la prolissità.
`-z`	comprime i dati durante il trasferimento.
`-e`	specifica la shell remota da usare.[7]
`-A`	preserva le ACL.
`-X`	preserva gli attributi estesi.
`-H`	riconosce gli *hard link* e li mantiene nella destinazione (dispendiosa).
`--exclude`	esclude un insieme di file dalla copia.
`--delete`	cancella sulla destinazione quanto non presente nella sorgente.

Tabella 4.7: Principali opzioni del comando `rsync`.

4.2 La gestione dei pacchetti software

Affronteremo in questa sezione le varie modalità in cui un amministratore può installare e rimuovere software dal sistema, in particolare esaminando i diversi sistemi di gestione dei pacchetti messi a disposizione dalle varie distribuzioni che permettono di tenere traccia, aggiornare e rimuovere tutto quello che si è installato nella propria macchina.

4.2.1 L'installazione diretta dai sorgenti

Una delle caratteristiche costitutive del software libero è la disponibilità del sorgente dei programmi, pertanto è sempre possibile effettuare una *installazione diretta* a partire dal codice che viene distribuito direttamente dagli sviluppatori. In generale questo può avvenire nelle modalità

[7] il default è `rsh`, che sui sistemi recenti è un alias a `ssh`, ma l'opzione consente di specificare l'intera riga di comando da invocare sull'altro lati della connessione.

più disparate, e dipende anzitutto dal linguaggio in cui lo stesso software viene scritto. Tratteremo in questa sezione il caso più comune, in cui i programmi da installare devono essere prima compilati, qualora il software sia stato scritto in un linguaggio interpretato in genere tutto quello che servirà sarà semplicemente installare i file che lo compongono in maniera opportuna nella propria macchina.

Nella maggior parte dei casi il software viene distribuito in forma sorgente in degli archivi compressi chiamati gergalmente "*tarball*", in quanto creati con il programma `tar`, trattato in sez. 4.1.2. In genere gli archivi vengono pure compressi, per cui di norma li si trovano distribuiti in file che usano l'estensione `.tar.gz` (talvolta abbreviata in `tgz`, per l'uso di sistemi obsoleti che hanno problemi coi nomi di file che hanno troppi caratteri "`.`" al loro interno) dato che il programma di compressione più usato è `gzip`. Una alternativa che si inizia a diffondere per gli archivi più grandi è quella dell'uso di `bzip2`, che comprime molto di più, anche se è più lento, nel qual caso l'estensione usata è `.tar.bz2`.

Per installare un pacchetto dai sorgenti la prima cosa da fare è scaricare l'archivio degli stessi dal sito di sviluppo: è consigliato usare, se ci sono, degli eventuali *mirror*, dato che questi di norma sono meno *occupati* del sito originale, e probabilmente anche più "*vicini*" in termini di distanza sulla rete e quindi con una maggiore velocità di accesso.

Una volta scaricato il pacchetto se ne può verificare il contenuto o scompattarlo usando il comando `tar` usando la sintassi trattata in sez. 4.1.2. Ad esempio se il pacchetto è distribuito nel file `pacchetto.tar.gz` lo si potrà decomprimere con il comando `tar -xvzf pacchetto.tar.gz`. In genere questo crea una directory identica a quella che conteneva tutti i file necessari sulla macchina in cui è stato creato l'archivio. È pratica comune usare una directory con lo stesso nome del pacchetto, a cui si suole aggiungere un numero di versione progressivo che consenta di distinguere i vari rilasci.

A questo punto per l'installazione occorre eseguire le operazioni previste allo scopo dagli sviluppatori. Queste possono risultare molto diverse da pacchetto a pacchetto, ma in genere chiunque distribuisce software fornisce anche le informazioni necessarie su come effettuare l'installazione. Di norma queste vengono distribuite insieme ai sorgenti, ed è prassi comune trovarle nei file `README` o `INSTALL`.

Si tenga presente comunque che installare un pacchetto dai sorgenti presenta dei requisiti maggiori rispetto all'installazione di un pacchetto binario, in quanto non basta copiare dei file nella loro destinazione finale. Infatti se il codice deve essere compilato sono necessari una serie di programmi che tipicamente non vengono installati in macchine su cui non si fa sviluppo, come compilatori, programmi per la costruzione dei binari, file di sviluppo delle librerie. Anzi è buona norma di sicurezza non avere installati compilatori ed altri tool di sviluppo su server di produzione, per rendere più complicata la vita ad eventuali "*intrusi*" che volessero installare i "*loro*".

Inoltre molti programmi per funzionare possono necessitare di altri programmi o, caso molto comune, fare uso di specifiche librerie di funzioni. Essendo questi per lo più sviluppati in maniera indipendente da altri progetti, normalmente non vengono distribuiti insieme ai sorgenti del programma, e quindi non è detto che siano già presenti ed installati. Ci sono cioè quelle che tecnicamente vengono chiamate *dipendenze*, che devono essere risolte prima di poter procedere all'installazione.

Per cercare di semplificare la procedura di installazione dai sorgenti il progetto GNU ha realizzato un insieme di programmi, chiamati *autotool*, che consente di verificare che tutto quello

che serve per la costruzione del pacchetto sia presente. Se un progetto usa gli *autotool*, o se segue la procedura di installazione definita da questi ultimi, tutto quello che c'è da fare è di entrare nella directory dove si è scompattato il contenuto della *tarball* ed eseguire la sequenza di comandi:

```
./configure
make
make install
```

Il primo comando, `./configure`, esegue uno script di shell fornito coi sorgenti che verifica che nel sistema sia presente tutto quello che serve per compilare il pacchetto. Questo è il punto più critico della procedura, in quanto se manca qualcosa lo script si interromperà dicendo o che non ci sono i vari programmi di compilazione e costruzione, o che non esiste il supporto per una funzionalità necessaria al programma o che mancano i file di dichiarazione relativi ad una certa libreria che serve al programma, o la libreria stessa.

Lo script `configure` fornito dagli *autotools* prevede anche che sia possibile controllare le modalità di compilazione attraverso una serie di variabili di ambiente, riportate in tab. 4.8, che permettono di modificare il compilatore e le relative opzioni. In generale lo script è in grado di riconoscere automaticamente i valori ottimali per le suddette variabili, e non è necessario utilizzarle.

Variabile	**Utilizzo**
`CC`	specifica un altro compilatore.
`CFLAGS`	specifica opzioni di compilazione addizionali.
`CPPFLAGS`	specifica opzioni addizionali per il preprocessore.
`LDFLAGS`	specifica opzioni per il linker.
`LIBS`	specifica librerie addizionali.

Tabella 4.8: Le variabili di ambiente usate dallo script di configurazione degli *Autotools*.

Per effettuare una installazione dei sorgenti il requisito fondamentale è ovviamente quello di avere installato il relativo compilatore. Nei casi più comuni i programmi sono scritti in linguaggi come il C o il C++ e allora è sufficiente installare i programmi `gcc` e `g++` e tutto quanto fa parte della *GNU Compiler Collection* che oltre a questi due compilatori contiene pure una serie di altri programmi accessori. In genere questi programmi vengono forniti direttamente dalla propria distribuzione, e si dovrà semplicemente installarli con le modalità previste da quest'ultima.[8]

Un secondo programma essenziale per l'installazione dai sorgenti è `make`, che, come dice il nome, viene usato per automatizzare la *costruzione* dei binari a partire dai sorgenti, consentendo di definire tutte le procedure necessarie per la creazione dei file nella giusta sequenza, sulla base di una serie di ricette contenute negli appositi `Makefile`. Il programma funziona eseguendo le istruzioni che trova in un file con questo nome posto nella directory corrente, è compito proprio dello script `configure` creare detto file con i corretti contenuti.

Se si è installato tutto quello che è necessario alla compilazione, e si sono risolte tutte le dipendenze da altro software o librerie, lo script `configure` terminerà senza errori e si potrà passare al comando successivo. In questa fase uno degli errori più comuni che si può presentare è che, pur avendo installato il software e tutte le librerie necessarie, non siano presenti i file

[8]nel caso di Debian si può installare in un colpo solo tutto il necessario alla compilazione di programmi di base richiedendo il meta-pacchetto `build-essential`.

di dichiarazione associati alla librerie, che sono necessari al compilatore per poter accedere alle funzioni in esse definite nella fase di compilazione.

Il problema è comune perché per un programma binario già compilato questi file non sono necessari, e pertanto è normale che non vengano installati anche quando ci sono le librerie corrispondenti. Però essi vengono messi a disposizione da tutte le distribuzioni come pacchetti con lo stesso nome di quello della libreria a cui fanno riferimento, a cui viene aggiunto un `-dev` per indicare che si tratta della parte necessaria allo sviluppo, servono infatti non solo per compilare un programma per l'installazione, ma anche qualora si volessero sviluppare programmi che utilizzano le corrispondenti librerie. In questo caso non c'è altro da fare che installare anche questi pacchetti di sviluppo, rifacendosi di nuovo al meccanismo di gestione fornito dalla propria distribuzione.

Una volta che `./configure` ha completato con successo le sue operazioni si potrà eseguire il secondo comando nella sequenza, `make`, che esegue la costruzione dei binari. Questo in genere dà luogo ad una lunga serie di messaggi da parte del compilatore, fino a quando le operazioni non sono concluse. Se qualcosa va storto in questa fase restano solo due strade, si può scrivere un *bug report* a chi ha creato il pacchetto (evitando troppi accidenti e fornendo le righe in cui si è verificato l'errore) ed aspettare una risposta o provare a correggere l'errore da soli, ma in quest'ultimo caso dovrete essere un po' pratici di programmazione, nel qual caso probabilmente potete evitare di leggere queste pagine. Come ultima risorsa potete comunque provare a cercare sui motori di ricerca se qualcuno ha già incontrato un problema simile al vostro e se ne è stata fornita una qualche soluzione.

Una volta completata la compilazione con `make install` si dice allo stesso programma di eseguire le istruzioni di installazione. Si tenga presente che la procedura standard, seguendo le linee guida del *Filesystem Hierarchy Standard* di sez. 1.2.3, prevede di installare i pacchetti compilati dai sorgenti sotto `/usr/local/`, per cui per poterli utilizzare occorre avere le directory di questa gerarchia secondaria nel PATH della shell (vedi sez. 2.1.3) e si deve essere in grado di usare le eventuali librerie ivi installate dal pacchetto (secondo quanto illustrato in sez. 3.1.2).

Se si usa la versione di `configure` degli *Autotool GNU* è possibile cambiare questa directory passando allo script `./configure` l'opzione `--prefix` che prende come parametro il pathname della directory dove si vuole che venga eseguita l'installazione. Alternativamente, per non dover rieseguire lo script di configurazione se ci si è dimenticati l'opzione, si può modificare la variabile `PREFIX` definita nelle prime righe del `Makefile` principale nella directory dei sorgenti. Si può anche indicare una specifica directory per l'installazione dei binari con l'opzione `--exec-prefix`, il cui default è comunque usare il valore indicato per `--prefix`.

Si tenga presente che la procedura descritta vale solo se gli sviluppatori hanno deciso di usare il sistema di costruzione e controllo fornito dagli *autotool* del progetto GNU. In generale non è detto che questo avvenga, nel qual caso occorre rifarsi alle istruzioni lasciate dal programmatore. Alternative possibili sono, per progetti non troppo complessi, la semplice invocazione del comando `make` (e di un successivo `make install`), o l'uso diretto del compilatore ma questo in genere solo in casi elementari.

Benché sia sempre possibile eseguire una installazione dai sorgenti, in una pratica di buona amministrazione questo dovrebbe essere evitato il più possibile anche quando, come avviene con la procedura appena illustrata, una tale installazione non risulti neanche particolarmente complessa. Infatti a parte i problemi preliminari relativi a dover sistemare a mano tutte le

dipendenze di un pacchetto, o al tempo impiegato per eseguire la compilazione dei programmi,[9] il problema maggiore resta la manutenzione di quanto si è installato, a causa delle difficoltà relative all'aggiornamento e alla eventuale rimozione.

Infatti se installare è facile, disinstallare è molto più problematico. Alcuni pacchetti (una minoranza purtroppo) forniscono uno speciale *bersaglio*[10] per `make` che consente la disinstallazione con `make unistall`. Ma se questo non c'è occorre tracciarsi a mano i file che sono stati installati e cancellarli. In caso di aggiornamento poi la cosa si complica ancora di più perché installando una nuova versione potrà essere necessario sovrascrivere file, o installarne di nuovi e cancellarne di vecchi, tutte cose che non sono quasi mai previste dalla procedura di installazione di un pacchetto.

Per questi motivi, come vedremo nelle sezioni seguenti, la gran parte delle distribuzioni fornisce un sistema di gestione dei pacchetti che permetta di tenere traccia di cosa si installa e di dove sono messi i file, che poi consente sia una cancellazione pulita del pacchetto quando lo si vuole disinstallare, che un aggiornamento corretto di tutto quello che serve nell'uso di nuove versioni.

4.2.2 La gestione dei pacchetti con `rpm` e `yum`

Uno dei primi sistemi evoluti per la gestione dell'installazione e rimozione di pacchetti software è RPM, realizzato da RedHat per la sua distribuzione. La sigla originariamente stava per *RedHat Package Manager*, essendo utilizzato da molte altre distribuzioni oggi viene chiamato semplicemente *RPM Package Manager*. Data la sua efficacia esso è stato in seguito adottato da molte altre distribuzioni, diventando uno dei sistemi di gestione dei pacchetti più diffusi.

Con RPM si intende sia il programma `rpm`, che permette di eseguire l'installazione e la rimozione dei pacchetti, che il formato dei file con cui sono distribuiti i pacchetti, normalmente caratterizzati dalla estensione `.rpm`. In sostanza si tratta di archivi `cpio` dei file necessari, tanto che esiste il comando `rpm2cpio` che consente di estrarre l'archivio `cpio` da un file `.rpm`.

In realtà il comando `rpm` fa molto di più che installare e disinstallare: mantiene la lista dei pacchetti installati e di dove sono stati installati i singoli file che ne fanno parte; così può accorgersi se un pacchetto va in conflitto con altri cercando di installare gli stessi file. Inoltre è in grado di accorgersi di dipendenze fra i vari pacchetti, di effettuare verifiche su quanto è stato installato e di eseguire degli script in fase di installazione e disinstallazione con cui eseguire procedure di configurazione automatica del pacchetto stesso.

Il comando è complesso e prevede diverse modalità operative, attivate da altrettante opzioni che non possono essere usate contemporaneamente. Le due modalità più usate sono quella di installazione, attivata dall'opzione `-i` e quella di rimozione, attivata dall'opzione `-e`. Altre modalità usate spesso sono quella di interrogazione del database dei pacchetti, attivata dall'opzione

[9] questo per pacchetti piccoli può anche essere trascurabile, ma per pacchetti importanti come il server X può essere, a seconda della potenza della macchina, dell'ordine delle ore o dei giorni.

[10] si chiamano così, dall'inglese *target*, gli argomenti che si passano a `make` e che normalmente indicano quali delle varie sezioni dei comandi di costruzione e installazione scritti in un `Makefile` devono essere usati, non specificare nulla significa usare il primo *target* disponibile che di norma esegue il compito principale; di solito ne esistono degli altri, ad esempio è comune `clean` che serve a cancellare i risultati della compilazione per riportare la directory del pacchetto nelle condizioni iniziali precedenti alla costruzione dello stesso, per ulteriori dettagli sul funzionamento di `make` si consulti [Make].

`-q` e quella di verifica attivata dall'opzione `-V`. Per l'elenco completo si consulti la pagina di manuale.

Nel caso si debbano installare dei pacchetti RPM occorre anzitutto procurarsi i file `.rpm` con il relativo contenuto, l'opzione `-i` (o `--install`) di `rpm` infatti prende come argomento il file o la lista di file dei nuovi pacchetti che si vogliono installare; alternativamente si può usare negli argomenti una URL al posto del nome di un file, nel qual caso il pacchetto indicato verrà anche scaricato direttamente dalla rete via HTTP o FTP.

I pacchetti RPM seguono anche una convenzione precisa per i nomi dei file; questi sono composti da più parti separate dal carattere "-". La prima parte indica il nome del pacchetto stesso, e può essere anche composta di più parole; segue il numero di versione, così come preso dalla versione originale dal software, cui segue il cosiddetto numero di *revisione* che marca le diverse versioni del pacchetto stesso. Infine segue, separate da un punto, l'architettura per cui il pacchetto è stato realizzato e l'estensione; in sostanza si avrà un nome del tipo:

```
logrotate-3.7.4-8.i386.rpm
```

Nell'installare un pacchetto `rpm` controlla preliminarmente che tutte le sue *dipendenze* siano soddisfatte, non installandolo in caso contrario ed indicando dove sta il problema. Il comando si rifiuta di installare il pacchetto anche se un file in esso contenuto esiste di già o se un altro pacchetto con lo stesso nome è già installato. Questo significa che non si può usare l'opzione `-i` per eseguire un aggiornamento, essa deve essere usata soltanto per installare pacchetti *nuovi*. Per eseguire un aggiornamento sono previste due opzioni alternative: `-U` (o `--upgrade`) con cui si richiede l'installazione di nuovi pacchetti e l'aggiornamento di eventuali pacchetti già installati, e `-F` (o `--freshen`) con cui si richiede il solo aggiornamento di pacchetti già installati.

Si tenga presente che oltre a gestire le dipendenze, `rpm` è in grado di gestire anche gli eventuali *conflitti*, cioè i casi in cui due pacchetti (o due versioni diverse dello stesso pacchetto) cercano di installare lo stesso file, ed in tal caso ovviamente dovrà essere generato un errore. Esistono casi però in cui, ad esempio a causa di un aggiornamento ad una versione di un pacchetto malfunzionante, può essere necessario evitare questi controlli.

A tale scopo sono state fornite alcune opzioni di controllo da aggiungere a `-i` per forzare l'installazione, come `--replacefiles`, che consente di installare il pacchetto anche se i file già sono presenti, rimpiazzandoli, `--replacepkgs` che consente di installare i pacchetti anche se alcuni di essi sono già presenti e `--oldpackage` che consente di effettuare un aggiornamento sostituendo una versione più recente dello stesso pacchetto con una precedente. É poi disponibile l'opzione `--force` che è una abbreviazione per l'uso di tutte e tre le opzioni precedenti.

Come accennato la disinstallazione di un pacchetto viene eseguita con l'opzione `-e` (o `--erase`) cui segue il nome del pacchetto o della lista dei pacchetti da rimuovere. Anche in questo caso vengono fatti dei controlli preliminari, e ad esempio l'azione viene abortita se un altro pacchetto dipende da quello che si vuole eliminare. Di nuovo può essere necessario forzare la disinstallazione, cosa che può essere fatta specificando la ulteriore opzione `--nodeps`, che consente di ignorare i problemi dovuti alle dipendenze (la stessa opzione è accettata anche da `-i`).

Con l'opzione `-q` (o `--query`) è possibile eseguire una serie di interrogazioni sul database dei pacchetti installati; se non si specifica altro l'opzione richiede come argomento un nome di pacchetto di cui verifica la presenza nel sistema. Ad essa però sono associate una lunga serie di altre opzioni che consentono di ottenere informazioni più specifiche; queste sono a loro volta

suddivise in quelle che indicano quali informazioni mostrare ed in quelle che indicano a chi applicare la richiesta.

Ad esempio con `rpm -qa` si ottiene la lista di tutti i pacchetti installati nel sistema. Se invece si usa `-ql`, seguito dal nome del pacchetto, si stampano i file in esso contenuti. Si può anche leggere il contenuto di un pacchetto non installato con `-qpl` dove `-p` serve a richiedere che come argomento si specifichi il file del pacchetto. Qualora invece si volesse sapere a quale pacchetto appartiene un file si potrà usare `-qf` indicando come argomento il file stesso.

La modalità di verifica viene attivata con l'opzione `-V` (o `--verify`) e richiede che si specifichi come argomento il nome di un pacchetto installato o `-a` per eseguire la verifica su tutti quanti. L'opzione consente di controllare l'integrità dei file che sono stati installati da un pacchetto rispetto alle informazioni sullo stesso mantenute nel database controllando se permessi, dimensioni e contenuto del file sono coincidenti con quelli previsti. L'opzione provvede l'uso di ulteriori opzioni per selezionare quali verifiche effettuare e restituisce per ciascun file una stringa con i risultati delle verifiche.

Una altra opzione che consente di verificare il contenuto dei pacchetti è `--checksig`, che serve a controllare originalità e integrità di un file contenente un pacchetto RPM, verificandone la firma digitale. Infine sono degne di nota le opzioni `--setperms` e `--setugids` che consentono di ripristinare ai valori originari permessi e proprietario di tutti i file di un pacchetto, qualora li si siano modificati per errore.

In tab. 4.9 si sono riportate le principali opzioni di `rpm`. La tabella è stata suddivisa in varie sezioni, accorpando insieme le opzioni relative alla varie modalità di operazione del comando, e lasciando nell'ultima parte le opzioni generiche. Per ciascuna sezione si sono anche evidenziate, riportandole per prime, le opzioni obbligatorie che fanno riferimento a quella modalità di operazioni e che devono essere presenti per poter utilizzare le altre.

A lungo `rpm` è stato in grado di gestire le dipendenze solo in forma molto elementare, avvisando l'utente di quali erano i file che dovevano essere già presenti nel sistema perché un certo pacchetto potesse essere installato, con il non banale svantaggio di costringere l'amministratore ad indovinare quale potesse essere il pacchetto che li conteneva. Nelle versioni più recenti sono state però aggiunte funzionalità più evolute in grado di eseguire il controllo delle dipendenze a livello di pacchetto e non di singolo file, e sono state introdotte anche nuove opzioni di interrogazione da abbinare all'opzione `-q`, come `--whatprovides`, che indica quali pacchetti forniscono una certa funzionalità, e `--whatrequires` che indica tutti i pacchetti che ne richiedono un altro per funzionare.

In questo modo il programma permette di sapere quali sono i pacchetti che non funzioneranno a causa della mancanza di un qualche componente a loro essenziale che potrà essere fornito da altri pacchetti. Il meccanismo fornito da `rpm` però si limita a questo, sta all'amministratore trovare, scaricare ed installare i pacchetti mancanti, il che porta a quella situazione che viene chiamata *dependency hell*, quando un pacchetto dipende da un altro che a sua volta dipende da una altro che a sua volta... Tutto questo a meno che la propria distribuzione non fornisca un sistema di gestione dei pacchetti di più alto livello, come `yum` o la versione per RPM del sistema APT di Debian, che tratteremo nella prossima sezione.

A lungo infatti le distribuzioni basate su RPM sono state prive di un sistema di gestione dei pacchetti di alto livello presente invece su Debian e sulle sue derivate, che permettesse di gestire in maniera più semplice le dipendenze e gli aggiornamenti del software. Per questo ognuna ha finito per sviluppare un suo sistema diverso. Fra questi quello che ha avuto il maggior successo,

Opzione		Significato
-i	*--install*	installa un nuovo pacchetto.
-U	*--upgrade*	aggiorna o installa un pacchetto.
-F	*--freshen*	aggiorna un pacchetto già installato.
	--replacefiles	sovrascrive file di altri pacchetti.
	--replacepkgs	installa anche se il pacchetto è già installato.
	--oldpackage	installa una versione precedente a quella attuale.
	--force	riassume --replacefiles, --replacepkgs e --oldpackage.
-h	--hash	stampa una barra di progressione.
-e	*--erase*	rimuove un pacchetto.
	--nodeps	non verifica le dipendenze.
-q	*--query*	interroga il database per la presenza di un pacchetto.
-a	--all	stampa tutti i pacchetti installati.
-l	--list	stampa i file contenuti in un pacchetto.
-f	--file	stampa il pacchetto che contiene un file.
-p	--package	stampa il contenuto del pacchetto contenuto nel file.
	--whatrequires	indica quali pacchetti dipendono da una certa funzionalità.
	--whatprovides	indica quali pacchetti forniscono una certa funzionalità.
-V	--verify	verifica lo stato di un pacchetto.
-K	--checksig	verifica la firma digitale di un pacchetto.
-v		stampa più informazioni nell'esecuzione delle operazioni.
	--setperms	ripristina i permessi dei file del pacchetto.
	--setugids	ripristina proprietario e gruppo dei file del pacchetto.

Tabella 4.9: Principali opzioni del comando rpm suddivise per categorie.

e che si avvicina di più alle funzionalità presenti su Debian è yum, il cui nome sta per *Yellowdog Updater Modified* essendo stato creato inizialmente da Yellowdog, ed in seguito modificato ed adottato da altre distribuzioni, come RedHat.

Il concetto generale di yum, ripreso pari pari dall'APT di Debian, è quello di avere una o più fonti, i cosiddetti *repository*, su cui sono contenuti sia i pacchetti che le informazioni ad essi relative (come le dipendenze reciproche, le versioni, ecc.), organizzate in maniera opportuna. Facendo riferimento ad un *repository* si potrà allora conoscere quali pacchetti sono disponibili, quali sono necessari per poterne installare un altro o di quali pacchetti esiste una versione più recente, da usare come aggiornamento.

Operazione	Significato
install	installa un pacchetto.
check-update	verifica se sono disponibili aggiornamenti.
update	esegue l'aggiornamento dei pacchetti installati.
upgrade	esegue l'aggiornamento anche da una distribuzione precedente.
remove	rimuove un pacchetto.
erase	sinonimo di remove.
list	stampa un elenco dei pacchetti.
info	stampa informazioni descrittive sui pacchetti.
clean	cancella la cache dei pacchetti scaricati.
search	esegue una ricerca della stringa passata come argomento nelle descrizioni e nei nomi dei pacchetti.

Tabella 4.10: Principali operazioni di yum.

In genere yum viene invocato passandogli come primo argomento l'operazione che si vuole

eseguire, seguita da eventuali altri argomenti successivi. Se ad esempio si vuole installare un pacchetto sarà necessario usare l'operazione `install` e specificare poi il nome dello stesso. In questo caso basta il nome del pacchetto, usando dei *repository* non è necessario né il nome di un file né un numero di versione. Specificando più nomi verrà richiesta l'installazione di più pacchetti. Sarà cura poi di `yum` scaricare ed installare preventivamente a quelli richiesti anche gli ulteriori pacchetti aggiuntivi ritenuti necessari a soddisfare le dipendenze.

Viceversa per rimuovere un pacchetto (o più) si può utilizzare l'operazione `erase` (o l'omonima `remove`) seguita dal nome (o dai nomi) dei pacchetti da rimuovere, mentre l'aggiornamento viene effettuato con `update`, anche in questo caso si può passare una lista di nomi, nel qual caso saranno aggiornati solo i pacchetti indicati. A questa si aggiunge `upgrade` che tiene in considerazione anche gli aggiornamenti dai pacchetti installati da una distribuzione precedente l'attuale.

Infine con `list` si possono elencare tutti i pacchetti disponibili (con versione e *repository* su cui si trovano) mentre con `info` se ne possono ottenere le relative informazioni, entrambe le operazioni riconoscono come parametri aggiuntivi, oltre ad un elenco di nomi di pacchetti, per i quali è anche possibile usare le *wildcard* del *filename globbing*, le parole chiave `all` che elenca tutti i pacchetti, `available` che elenca quelli disponibili per l'installazione, `updates` che elenca gli aggiornamenti, `installed` che elenca quelli installati ed `extras` che elenca quelli installati ma non presenti in nessun *repository*.

Se invece di installare un pacchetto lo si volesse soltanto scaricare, `yum` non supporta direttamente l'operazione, ed occorre fare ricorso ad un comando ausiliario, `yumdownloader`, che prende come argomenti una lista di pacchetti e li scarica nella directory corrente. Il comando consente di specificare una directory diversa con `--destdir` e di richiedere, se disponibile, il corrispondente pacchetto sorgente con l'opzione `--source`.

Per il funzionamento di `yum` e dei relativi comandi ausiliari occorre però poter fare riferimento agli opportuni *repository* che contengono i pacchetti ed il comando potrà accedere al contenuto degli stessi per ottenere automaticamente tutto quello che serve. L'accesso ad un *repository* può essere eseguito in una molteplicità di modi, ma i principali sono via rete con i protocolli HTTP o FTP, o direttamente tramite il contenuto, opportunamente strutturato, di una directory del proprio disco.

Il file di configurazione principale di `yum` è `/etc/yum.conf`, ma il programma riconosce anche `/etc/yum/yum.conf`. Questo file usa il formato dei file INI di Windows, che prevede delle sezioni introdotte da un nome posto fra parentesi quadre, all'interno delle quali si possono impostare le varie opzioni di configurazione nella forma di assegnazione di un valore ad una parola chiave. Un esempio di questo file, preso dalla versione installata su una Scientific Linux 6.2, è il seguente:

/etc/yum.conf

```
[main]
cachedir=/var/cache/yum/$basearch/$releasever
keepcache=0
debuglevel=2
logfile=/var/log/yum.log
exactarch=1
obsoletes=1
gpgcheck=1
plugins=1
installonly_limit=3
...
```

Il file richiede sempre la presenza di una (ed una sola) sezione `main`, che contiene le configurazioni generali del comando; è qui ad esempio che si può impostare, qualora sia necessario, l'utilizzo di un proxy web per il download dei pacchetti, assegnando all'opzione `proxy` la URL dello stesso (utilizzando poi le ulteriori `proxy_username` e `proxy_password` per indicare eventuali credenziali di accesso).

Opzione	**Significato**
`cachedir`	imposta la directory dove sono mantenuti i pacchetti scaricati e le relative informazioni.
`logfile`	imposta il file con il registro delle operazioni effettuate da `yum` (installazioni, aggiornamenti, rimozioni, ecc.).
`gpgcheck`	richiede o meno il controllo della firma con GPG dei pacchetti (rispettivamente con i valori "1" o "0").
`pkgpolicy`	imposta la politica da seguire quando lo stesso pacchetto è disponibile su diversi repository, con `newest` installa sempre la versione più recente, con `last` installa la versione presente nel *repository* che viene per ultimo nella lista degli stessi fatta per ordine alfabetico sul relativo identificatore.
`proxy`	imposta l'uso di un *proxy*, da specificare tramite la relativa URL.
`proxy_password`	imposta una password per l'accesso al *proxy*.
`proxy_username`	imposta un utente per l'accesso al *proxy*.
`reposdir`	imposta la directory (o le directory) da cui prendere la lista dei file `.repo` che definiscono i *repository*, il default è `/etc/yum.repos.d/`.
`tolerant`	rende il comando più tollerante (rispettivamente con i valori "1" o "0") rispetto ad alcune classi di errori (come richiedere l'installazione di un pacchetto già installato).

Tabella 4.11: Principali opzioni della sezione `main` di `yum.conf`.

Altre opzioni comuni, presenti anche nel precedente esempio, sono `cachedir` che imposta la directory dove verranno mantenuti i pacchetti scaricati ed i dati ad essi associati, e `logfile` che indica il file su cui vengono registrati i risultati delle varie operazioni. Oltre a queste sono definite una grande quantità di altre opzioni che consentono di controllare nei dettagli i vari aspetti di funzionamento del comando, le principali sono riportate in tab. 4.11 ma per i dettagli al solito occorre fare riferimento alla pagina di manuale, accessibile con `man yum.conf`.

Benché sia possibile indicare direttamente all'interno di `yum.conf` i *repository* che si vogliono utilizzare, definendo per ciascuno di essi altrettante sezioni, normalmente viene definita solo la sezione `main` e si preferisce usare a tale scopo una serie di file separati, caratterizzati dall'estensione `.repo`, che di default sono mantenuti nella directory `/etc/yum.repos.d/` (ma se può indicare un'altra con il valore della direttiva `reposdir` di `yum.conf`). In questo modo per aggiungere un nuovo *repository* è sufficiente installare un altro file all'interno di essa.

Il formato di un file `.repo` prevede che esso sia nella forma di una o più sezioni associate ad altrettanti *repository*. Il nome di ciascuna sezione, nella forma di una singola parola, deve essere sempre diverso in quanto viene a costituire il cosiddetto *repository id* che serve come riferimento univoco per identificare uno specifico *repository*. Ad esempio nel caso di CentOS 5.2 viene usato un file `CentOS-Base.repo` che definisce le sezioni `base`, `updates`, `addons`, `extras` e `centosplus` per l'uso dei *repository* ufficiali supportati da questa distribuzione, corrispondenti rispettivamente ai

pacchetti di base, agli aggiornamenti, e ad una serie di diverse estensioni e pacchetti aggiuntivi, ma in generale ogni distribuzione fornirà i suoi *repository* ufficiali.

Opzione	**Significato**
`name`	una stringa di descrizione del *repository*.
`baseurl`	la URL che indica l'indirizzo a cui è possibile raggiungere i contenuti del *repository*.
`mirrorlist`	si può usare al posto di `baseurl` per indicare un file che contiene una lista di URL.
`gpgkey`	la URL da cui scaricare la chiave pubblica GPG per la verifica di autenticità del pacchetto.
`exclude`	una lista, separata da spazi, di pacchetti da non prendere in considerazione anche se presenti, supporta anche l'uso dei caratteri jolly del *filename globbing*.
`enabled`	consente di disabilitare il *repository* se posto a 0, il valore di default è 1.

Tabella 4.12: Principali opzioni usabili per le sezioni che identificano un *repository*.

Ciascuna sezione dovrà poi contenere le opzioni che consentono di utilizzare il *repository* da essa identificato, la principale di queste è `baseurl`, che prende come argomento la URL a cui è possibile raggiungere i contenuti del repository. Le forme supportate sono `http://` per l'accesso HTTP, `ftp://` per l'accesso FTP e `file://` per l'accesso su una directory locale. Oltre a questa si sono riportate le altre opzioni principali in tab. 4.12; come al solito per i dettagli si consulti la pagina di manuale.

4.2.3 Il sistema di gestione dei pacchetti APT

Uno dei punti di forza di Debian è il sistema di gestione dei pacchetti creato da questa distribuzione; il sistema si chiama APT, sigla che sta per *Advanced Package Tool*. Anche in questo caso c'è un programma di base per la gestione dei singoli pacchetti, con funzionalità analoghe a quelle di `rpm`, questo però è solo una parte di una infrastruttura molto più ampia, che consente una gestione semplice e funzionale dell'installazione di pacchetti software.

I pacchetti Debian sono distribuiti in file con l'estensione `.deb` ed anch'essi seguono una precisa convenzione relativa ai nomi che in questo caso sono composti da più parti tutte separate dal carattere "_". La prima parte indica il nome del pacchetto stesso, e può essere anche composta di più parole. Segue la versione del pacchetto che è composta da due parti separate da un "`-`": il numero di versione del software originale, ed il numero di *revisione* che marca le diverse versioni del pacchetto stesso. Infine segue l'architettura per cui il pacchetto è stato realizzato e l'estensione, in sostanza si avrà un nome del tipo:

```
logrotate_3.7.1-3_i386.deb
```

Il programma di base analogo a `rpm` è `dpkg`, anche se in realtà `dpkg` è nato prima di `rpm`, per cui forse sarebbe più corretto dire che `rpm` è analogo a `dpkg`. Il comando fornisce le stesse funzionalità: installa e rimuove pacchetti mantenendo un database dei pacchetti e dei relativi file installati, provvede all'esecuzione di script in fase di installazione e rimozione, è in grado di accorgersi di eventuali conflitti con pacchetti già installati, e delle dipendenze da altri pacchetti, può interrogare il database dei pacchetti e ottenere informazioni sul loro contenuto.

Le opzioni principali del comando sono `-i`, che esegue l'installazione di un file `.deb`, e `-r`, che rimuove un pacchetto dal sistema. Con `-l` se non si specifica nulla viene stampata la lista dei pacchetti installati, mentre se si specifica un nome viene ricercata la presenza di un pacchetto con quel nome. Con `-L` si può stampare la lista dei file contenuti nel pacchetto e con `-S` quella dei pacchetti che contengono un file corrispondente alla stringa passata come parametro. Le altre opzioni principali sono riportate in tab. 4.13. Al solito le istruzioni complete e tutte le altre opzioni sono descritte nella pagina di manuale.

Opzione	**Significato**
`-i`	installa il pacchetto.
`-r`	rimuove il pacchetto.
`-l`	interroga il database per la presenza di un pacchetto.
`-L`	stampa i file contenuti in un pacchetto.
`-S`	ricerca i pacchetti che contengono un file.
`-s`	stampa lo *stato* di un pacchetto.
`-p`	stampa informazioni su un pacchetto installato.
`-I`	stampa informazioni su un `.deb`.
`-c`	stampa il contenuto di un `.deb`.

Tabella 4.13: Principali opzioni del comando `dpkg`.

Benché ormai anche i `.deb` risultino piuttosto usati, dato che sono diventate parecchie le distribuzioni basate su Debian, la loro diffusione su Internet come file singoli è piuttosto ridotta. Questo avviene proprio perché praticamente da sempre `dpkg` costituisce soltanto la parte di basso livello del sistema di gestione dei pacchetti di Debian, per cui è piuttosto raro dover usare questo comando per installare un pacchetto.

Come accennato infatti uno dei grandi vantaggi di questa distribuzione, e di quelle che usano lo stesso sistema di gestione, è costituito dalla interfaccia di alto livello costituita da APT, che permette di dimenticarsi completamente del problema delle dipendenze. I pacchetti Debian infatti sono stati organizzati fin dall'inizio per indicare in maniera coerente da quali altri pacchetti essi dipendono. Diventa così possibile richiedere l'installazione automatica non solo di un singolo pacchetto, ma anche di tutti quelli da cui questo dipende. Inoltre in genere i pacchetti vengono distribuiti direttamente via rete, con una modalità che permette il download automatizzato degli stessi.

Il programma di gestione principale per i pacchetti Debian non è allora `dpkg`, ma `apt-get`, che serve appunto da front-end per tutto il sistema dell'*Advanced Package Tool*. Ad esso si è aggiunto, ed è indicato come nuovo default, il comando `aptitude`, che oltre a funzionare a riga di comando fornisce anche una interfaccia testuale semi-grafica. Dato che `apt-get` fornisce in sostanza un sottoinsieme delle funzionalità di `aptitude`, parleremo prima di lui.

Per il corretto funzionamento del sistema di APT è fondamentale mantenere una lista delle necessarie fonti dei pacchetti (i *repository*) nel file `/etc/apt/sources.list`. Le versioni più recenti di APT prevedono anche l'uso della directory `/etc/apt/sources.list.d/` dove si possono indicare fonti aggiuntive; in tal caso è sufficiente inserire in questa directory un file con l'estensione `.list` e lo stesso formato di `sources.list`, ed il suo contenuto sarà utilizzato come se lo si fosse incluso in quest'ultimo.

Il formato del file `sources.list` prevede una serie di righe formate da campi divisi da spazi, ciascuna delle quali indica una fonte di pacchetti. Il primo campo indica il tipo di pacchetti

presenti in quel *repository*; originariamente i soli valori possibili erano `deb` e `deb-src`, per indicare rispettivamente pacchetti binari e sorgenti. Il sistema di APT però è stato portato anche su distribuzioni basate su RPM, sulle quali è possibile utilizzare per questo campo il valore `rpm`.

Il secondo campo è una URI per l'accesso al *repository* che indica la base del rispettivo archivio. Come accennato l'accesso può essere effettuato in modalità diverse, le principali delle quali sono indicate con le URI riportate in tab. 4.14, a cui dovrà seguire la opportuna specificazione della posizione del *repository*.

URI	**Significato**
`file:`	accede ai pacchetti posti in una directory locale (o anche remota se si usa NFS, vedi sez. 8.4.2).
`cdrom:`	accede ai pacchetti su un CDROM, una voce di questo tipo deve essere creata con l'ausilio di `apt-cdrom`.
`http:`	un ordinario riferimento ad un sito web.
`ftp:`	un ordinario riferimento ad un sito FTP.

Tabella 4.14: Principali formati delle URI usate in `sources.list`.

I campi successivi prevedono due diverse sintassi. La prima è quella in cui si specifica in un unico campo il pathname della directory in cui si trovano i pacchetti, così come viene vista relativamente alla base specificata con la URI del secondo campo. In questo caso il campo deve essere obbligatoriamente terminato dal carattere "`/`" per distinguerlo dall'altra sintassi. Si usa in genere questa forma per dei *repository* elementari in cui tutti i pacchetti sono stati inseriti in una unica directory.

Nella seconda sintassi il terzo campo indica invece il nome associato alla versione della distribuzione che si sta usando. Per Debian è sempre possibile usare i nomi generici `stable`, `testing` ed `unstable` ad indicare rispettivamente la versione stabile, quella di test e quella di sviluppo; a ciascuno di questi viene anche associato un nome proprio, che per `unstable` è sempre `sid`, mentre per le altre varia da rilascio a rilascio.[11] A questo segue almeno un campo aggiuntivo che indica la *sezione* del *repository* a cui si vuole avere accesso; nel caso di Debian le sezioni sono tre e cioè `main`, `contrib` e `non-free`. Sia il nome della versione, che le sezioni in cui essa è divisa, dipendono dalla distribuzione; ad esempio con Ubuntu, al momento della stesura di queste note, la versione corrente è `lucid`, mentre le sezioni utilizzate sono `main`, `restricted`, `universe` e `multiverse`.

Quando si usa questa seconda sintassi, che è quella usata di default per i repository ufficiali dalle distribuzioni che usano APT, si fa riferimento a dei *repository* più strutturati, in cui i pacchetti sono opportunamente organizzati in opportune directory.[12] Si noti anche come in nessuno dei due casi si sia fatto riferimento alla architettura hardware per la quale sono compilati i pacchetti binari; con la prima sintassi questo avviene perché i *repository* specificati in quella forma non supportano l'uso di architetture multiple, per cui si deve avere cura di indicarne con pacchetti adatti alla propria. Nel secondo caso invece non è necessario perché la scelta dei pacchetti corretti viene fatta in maniera automatica.

In genere la configurazione di un *repository* si fa semplicemente aggiungendo le righe necessarie a `sources.list`, l'unico caso che necessita di una cura particolare è quello relativo ai

[11] i nomi vengono presi dai personaggi del film "*Toy Story*", al momento della scrittura di questo testo la versione stabile è `squeeze` mentre quella di test è `wheezy`.

[12] i dettagli di tutto ciò vanno oltre lo scopo di questo testo, gli interessati possono consultare il capitolo 4 di [DebSys].

pacchetti di un CDROM, in cui la riga di configurazione deve essere creata con il programma `apt-cdrom`, che si incarica di verificare il contenuto di un CD e generare una voce con le informazioni necessarie, che permette di evitare problemi qualora si cambiasse disco nel lettore.

In genere se si è installato il sistema da uno o più CD o DVD la relativa voce completa viene creata automaticamente in fase di installazione ed è necessario modificarla solo se si vogliono usare altri supporti. Per l'uso del comando si può consultare la pagina di manuale, ma in generale tutto quello che c'è da fare è inserire il CD nel lettore ed eseguire `apt-cdrom add`. In seguito quando verrà richiesta l'installazione di un pacchetto dal CD sarà lo stesso `apt-get` a richiederne l'inserimento nel lettore.

Si tenga presente che è possibile indicare anche diversi *repository*, sia per indicare fonti aggiuntive di pacchetti. Ad esempio nel caso di Debian oltre al *repository* dei pacchetti ordinari si usa indicare anche quello degli aggiornamenti, su cui vengono inseriti tutti i pacchetti rilasciati per chiudere bug di sicurezza, che è mantenuto centralmente dal *security team* del progetto. Si possono anche indicare più righe con i mirror degli stessi *repository* per poter scaricare da diverse fonti gli stessi pacchetti in caso una non fosse raggiungibile. In quest'ultimo caso si tenga conto che la scansione avviene in ordine sequenziale, per cui è opportuno mettere prima quelle più vicine. Un esempio del file `sources.list`, tratto dalla versione presente (con l'installazione via rete) su una Debian Squeeze, è il seguente:

/etc/apt/sources.list
```
# deb cdrom:[Debian GNU/Linux 6.0.0 _Squeeze_ - Official i386 NETINST Binary-1
 20110205-14:34]/ squeeze main

deb http://ftp.it.debian.org/debian/ squeeze main
deb-src http://ftp.it.debian.org/debian/ squeeze main

deb http://security.debian.org/ squeeze/updates main
deb-src http://security.debian.org/ squeeze/updates main

deb http://ftp.it.debian.org/debian/ squeeze-updates main
deb-src http://ftp.it.debian.org/debian/ squeeze-updates main
```

dove, sia per i pacchetti binari che per i sorgenti, è stato indicato un mirror italiano dei *repository* ufficiali di Debian, ed il *repository* speciale relativo agli aggiornamenti di sicurezza.

Una volta impostati i vari *repository* il comando che consente di operare sui pacchetti è, come accennato, `apt-get`. Questo prende sempre come primo argomento una parola chiave che indica l'operazione da eseguire, seguito da eventuali altri argomenti, quando previsti. Si tenga presente che in generale, per evitare pericolosi intrecci, è possibile eseguire una sola istanza alla volta di `apt-get` o di qualunque altro programma che operi sulla installazione dei pacchetti. Se pertanto è già in corso una operazione sui pacchetti `apt-get` si bloccherà notificando che il sistema di gestione di APT è in già in uso.

In generale il primo passo da fare per operare con `apt-get` è eseguire il comando `apt-get update` per scaricare una lista aggiornata di tutti i pacchetti disponibili. Sarà possibile poi installare un pacchetto qualunque con il comando `apt-get install nomepacchetto`, dove il nome del pacchetto corrisponde alla prima parte (quella che precede il primo "_") del nome del file che lo contiene. Il programma si incaricherà di effettuare automaticamente il download dello stesso e di quelli necessari per soddisfare eventuali dipendenze, per poi procedere ad installare

il tutto. Qualora si voglia reinstallare un pacchetto già installato, si dovrà aggiungere l'opzione `--reinstall`.

Il comando gestisce anche eventuali *conflitti*, se ad esempio si cerca di installare un pacchetto che fornisce in maniera alternativa lo stesso servizio di un altro occorrerà prima rimuovere quest'ultimo. In questo caso, ed anche tutte le volte che è necessario installare altri pacchetti a causa delle dipendenze, il comando si ferma indicando quali pacchetti intende installare e quali rimuovere, e chiede una conferma prima di procedere nelle sue operazioni.

Si tenga presente poi che oltre alla installazione, nel sistema di gestione dei pacchetti Debian è integrato anche un meccanismo per l'auto-configurazione dei programmi (chiamato *debconf*) che fornisce una interfaccia generica con cui richiedere all'amministratore le informazioni necessarie per generare automaticamente una configurazione corretta o notificargli eventuali azioni necessarie per completare la configurazione. Il meccanismo è realizzato dall'omonimo pacchetto `debconf`, che può essere opportunamente configurato per richiedere un quantitativo maggiore o minore di dettagli, fino all'estremo di non fare chiedere nulla e generare un default generico (probabilmente non funzionante); si tenga presente che il sistema è opzionale e non è detto che sia utilizzato da tutti i pacchetti.

Questo risulta molto utile in caso di installazione di programmi di servizio, dato che in questo modo APT non solo installa il pacchetto, ma lo configura e rende immediatamente attivo. Pertanto la procedura di installazione può venire sospesa in attesa di una risposta da parte dell'amministratore. Un'altra caratteristica di *debconf* è che può essere configurata la modalità con cui viene eseguita la richiesta, il default è utilizzare una interfaccia semi-grafica a terminale, ma esistono varie alternative.

Qualora si voglia rimuovere un pacchetto il comando è `apt-get remove nomepacchetto`, ed in questo caso, se altri pacchetti dipendono da quello che si vuole rimuovere, il comando chiede conferma della volontà di rimuovere anche loro, ed in caso affermativo procede alla rimozione anche di questi. Una delle caratteristiche di Debian è che la rimozione di un pacchetto non comporta normalmente la rimozione dei relativi file di configurazione, che potrebbero tornare utili in una successiva reinstallazione. Se si vuole rimuovere definitivamente anche tutte le configurazioni occorre aggiungere al precedente comando l'opzione `--purge`.

Infine il sistema di APT consente una estrema facilità di aggiornamento del sistema, basta infatti usare il comando `apt-get upgrade` dopo aver usato `apt-get update` per ottenere automaticamente l'aggiornamento a eventuali nuove versioni presenti sui *repository* di tutti i pacchetti che sono già installati nel sistema. Il comando però non esegue mai un aggiornamento se questo comporta l'installazione di pacchetti aggiuntivi o la rimozione di un pacchetto già presente dal sistema, anche quando questo verrebbe sostituito da una diversa versione più aggiornata.[13]

Per risolvere questo tipo di situazioni, che nell'uso di una versione stabile si incontrano soltanto quando si effettua l'aggiornamento ad una versione stabile successiva, si deve usare il comando `apt-get dist-upgrade` che esegue una risoluzione intelligente dei conflitti ed è in grado di effettuare l'aggiornamento dei pacchetti principali a scapito di quelli secondari, accorgersi delle sostituzioni di pacchetti, installare le parti che mancano, ecc.

[13]in genere il problema si presenta per librerie o per programmi che vengono suddivisi in più parti, in tal caso se cambia il numero di versione della libreria cambia il nome del relativo pacchetto, mentre può non cambiare il nome del pacchetto che la usa, oppure un pacchetto può essere spezzato in più parti e quello originario non esiste più e devono essere installate tutte le varie parti.

Qualora si usi una versione di test o di sviluppo, cioè `testing` o `unstable`, la situazione descritta invece è comune e capita spesso di dover usare `dist-upgrade`. In tal caso si ponga attenzione alle eventuali rimozioni proposte, che vengono sempre indicate dal comando, il quale prima di operare chiede conferma mostrando i pacchetti che installerà e quelli che rimuoverà. In caso di errori nelle dipendenze dei pacchetti, che nelle versioni di sviluppo si verificano con una certa frequenza, specie quando viene fatta una transizione a nuove versioni, si potrebbe rischiare di rimuovere parti essenziali per il funzionamento di altri pacchetti.

Operazione	**Significato**
`install`	installa un pacchetto.
`remove`	rimuove un pacchetto.
`update`	scarica la lista aggiornata dei pacchetti.
`upgrade`	esegue l'aggiornamento dei pacchetti installati.
`dist-upgrade`	esegue l'aggiornamento della distribuzione.
`clean`	cancella l'archivio dei pacchetti scaricati.
`download`	scarica il pacchetto nella directory corrente.
`autoclean`	cancella dall'archivio le vecchie versioni dei pacchetti scaricati.

Tabella 4.15: Principali operazioni dei comandi `apt-get` e `aptitude`.

Altre operazioni di `apt-get` riguardano la gestione dell'archivio dei pacchetti scaricati, che normalmente vengono mantenuti in una apposita directory (`/var/cache/apt/archive`) così che non ci sia da riscaricarli qualora si debbano reinstallare. Le principali operazioni di `apt-get` sono riportate in tab. 4.15, l'elenco completo, insieme alla descrizione dettagliata di tutte le caratteristiche del comando, è al solito disponibile nella relativa pagina di manuale.

Il comando supporta inoltre alcune opzioni generiche, oltre le già citate `--reinstall` e `--purge` che sono specifiche dell'operazione `install`. In particolare con `-d` si può richiedere che i pacchetti necessari per una operazione siano soltanto scaricati, mentre con `-f` è possibile richiedere un tentativo di correzione in caso di sistema in cui le dipendenze sono state spezzate, ad esempio da un aggiornamento interrotto. Per i dettagli si consulti al riguardo la pagina di manuale.

Le operazioni di tab. 4.15 sono utilizzate anche dal comando `aptitude`, una implementazione alternativa che per un certo tempo è stato indicato come il programma ufficialmente deputato alla gestione dei pacchetti con APT. Il suo principale vantaggio è quello di poter essere utilizzato in una interfaccia semigrafica a terminale, cui si accede invocandolo senza nessun argomento, in tal caso si otterrà una schermata come quella illustrata in fig. 4.1, da cui si potranno dare i vari comandi (ed utilizzare pure un help in linea).

In generale però `aptitude` può essere usato da riga di comando in maniera analoga ad `apt-get`, fintanto che si usano le operazioni di base illustrate in tab. 4.15. Il comando però non è totalmente equivalente, infatti le opzioni `--purge` e `--reinstall` non esistono e sono state trasformate in omonime operazioni separate. Inoltre le operazioni di aggiornamento, pur mantenendo per compatibilità lo stesso nome, sono svolte con algoritmi diversi e ad esempio con `upgrade` pacchetti non presenti possono essere installati per risolvere le dipendenze, e possono essere disinstallati pacchetti non utilizzati.

Inoltre il programma consente un maggiore controllo sugli aggiornamenti, ad esempio con l'operazione `hold` si può bloccare indefinitamente l'aggiornamento di un pacchetto, fintanto che questo non viene sbloccata con `unhold`. In tab. 4.16 si sono riportate alcune operazioni specifiche di `aptitude`, per tutti i dettagli (il comando è molto potente e supporta un gran numero di

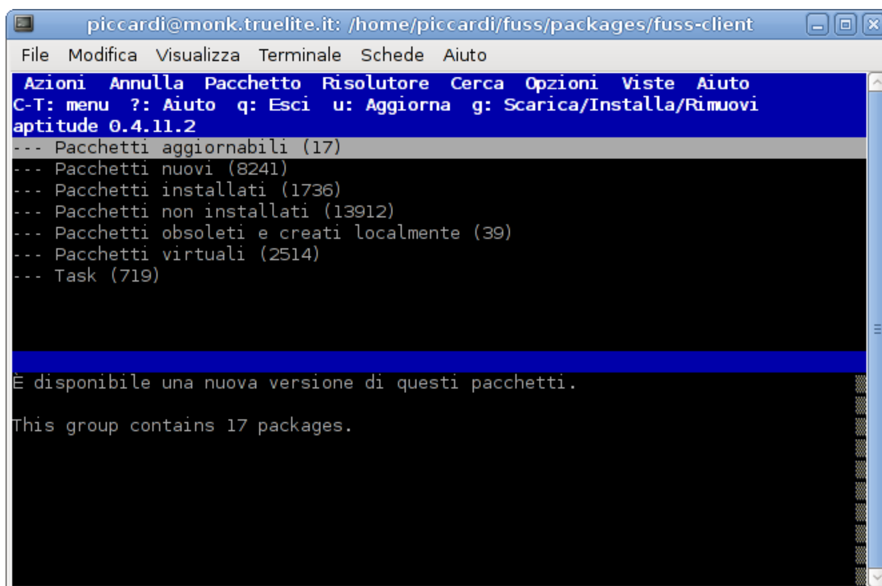


Figura 4.1: Schermata di avvio di `aptitude`.

operazioni) si può di nuovo consultare la pagina di manuale o il manuale utente, la cui versione testuale è disponibile nel file `/usr/share/doc/aptitude/README`.

Operazione	**Significato**
`hold`	blocca l'aggiornamento di un pacchetto.
`unhold`	sblocca l'aggiornamento di un pacchetto.
`purge`	rimuove un pacchetto e le sue configurazioni.
`reinstall`	forza la reinstallazione di un pacchetto.
`forbid-version`	evita l'aggiornamento per una certa versione.
`safe-upgrade`	equivalente di `upgrade`.
`full-upgrade`	equivalente di `dist-upgrade`.
`search`	esegue una ricerca sul database dei pacchetti.
`show`	mostra le informazioni su un pacchetto.

Tabella 4.16: Principali operazioni specifiche di `aptitude`.

Oltre alla gestione dell'installazione dei pacchetti `aptitude` consente anche di eseguire delle interrogazioni sul database dei pacchetti per ottenere delle informazioni riguardo gli stessi. In realtà a tale scopo è presente anche un comando dedicato, `apt-cache`, che presenta funzionalità più estese. Il funzionamento del programma è simile agli altri e prevede un primo argomento che indica l'operazione da eseguire.

Si può allora effettuare una ricerca di un testo qualunque nella descrizione dei pacchetti con il comando `apt-cache search testo`, in cui il secondo argomento costituisce il testo da cercare (in realtà l'argomento viene utilizzato come espressione regolare). Si possono in questo caso passare anche più argomenti successivi e verrà eseguita una ricerca sulla presenza di tutti quanti. Una seconda operazione comune è `apt-cache show pacchetto` che consente di ottenere le

caratteristiche del pacchetto, secondo quanto scritto nella sua descrizione, ad essa si affiancano altre operazioni, citate in tab. 4.17, che consentono di richiede informazioni sulle dipendenze.

Operazione	**Significato**
`search`	esegue una ricerca sulle descrizioni dei pacchetti con le espressioni regolari passate come argomenti.
`show`	mostra le informazioni relative alle caratteristiche del pacchetto passato come argomento.
`policy`	stampa le priorità associate a ciascun pacchetto passato come argomento ed i *repository* da cui esso può essere ottenuto.
`depends`	stampa la lista dei pacchetti da cui dipende il pacchetto passato come argomento (dipendenze dirette).
`rdepends`	stampa la lista dei pacchetti che dipendono dal pacchetto passato come argomento (dipendenze inverse).
`showpkg`	stampa la versione e tutte le informazioni riguardo dipendenze dirette ed inverse del pacchetto passato come argomento.

Tabella 4.17: Principali operazioni di `apt-cache`.

Le principali operazioni di `apt-cache` sono state riassunte in tab. 4.17, e come al solito si può ottenere l'elenco completo dalla pagina di manuale del comando. Fra quelle riportate vale la pena evidenziare `apt-cache policy`, che consente di ottenere le *priorità* delle varie versioni di un pacchetto e l'elenco dei repository da cui esso è disponibile, perché questo ci consente di introdurre una funzionalità molto utile di APT che viene utilizzata spesso.

Uno dei problemi che si può avere con una distribuzione è che i pacchetti distribuiti ufficialmente dalla stessa possono non essere molto aggiornati; questo è particolarmente comune con Debian che segue la politica di non aggiornare mai la versione dei pacchetti presenti nella distribuzione stabile,[14] se non per inserire correzioni di sicurezza, riportate indietro però sulla versione del pacchetto presente nella distribuzione. In questo modo si evita l'introduzione di eventuali nuovi errori o cambiamenti di comportamento che si potrebbero avere in una versione successiva, ottenendo il massimo della stabilità, a scapito però dell'uso di funzionalità presenti nelle versioni più recenti.

Esistono però alcune situazioni in cui sono essenziali delle funzionalità presenti soltanto nella versione più recente di un certo pacchetto; per questo è possibile fare riferimento ad altri *repository* oltre quelli ufficiali, da cui ottenere una versione aggiornata. In particolare per Debian è estremamente comodo il *repository* di `http://backports.debian.org` che contiene versioni più recenti di molti pacchetti ricompilati per essere usati nella versione stabile senza dover installare niente altro che il pacchetto stesso. In tal caso si tratta di aggiungere le opportune fonti in `sources.list`, e quello che succede è che APT installerà la versione più recente fra quella dei pacchetti disponibili. In situazioni come queste l'uso di `apt-cache policy` è d'aiuto per capire quale versione del pacchetto si installa e da quale *repository*.

Un ultimo aspetto importante del sistema di gestione dei pacchetti di Debian è che, come accennato, esso fornisce, con *debconf*, un meccanismo generico per l'auto-configurazione degli stessi. Questo non solo consente di richiedere (e memorizzare) le eventuali informazioni necessarie per

[14] la scelta non è solo di Debian, anche RedHat segue una politica simile con la sua RHEL, perché in ambito professionale la stabilità è un requisito fondamentale, e dover aggiornare le macchine ogni sei mesi è un grave problema, non un vantaggio.

la configurazione automatica attraverso una interfaccia generica, ma fornisce anche delle funzionalità aggiuntive, come il fatto che aggiornando un pacchetto possa essere automaticamente aggiornata anche la sua configurazione.

Questo avviene soltanto se non si sono fatte modifiche manuali alla configurazione originale, in caso contrario il sistema chiede cosa fare, consentendo anche di visualizzare le differenze fra nuova e vecchia configurazione; in questo modo è possibile effettuare una scelta, inoltre in caso di cambiamento viene sempre salvata una copia del precedente file di configurazione (come `.dpkg-old`) se si è deciso di installare il nuovo, o del nuovo (come `.dpkg-dist`) se si è deciso di mantenere il vecchio.

In generale le informazioni necessarie per la configurazione vengono richieste quando si installano i pacchetti ed i relativi file di configurazione vengono generati automaticamente, può capitare però che si voglia riconfigurare il pacchetto in un secondo tempo. A tale scopo viene fornito il comando `dpkg-reconfigure` che consente di ripetere questa procedura, rieseguendo anche la richiesta delle eventuali informazioni e rigenerando la configurazione.

Il comando prende come argomento il pacchetto o i pacchetti che si vogliono riconfigurare, o l'opzione `-a` qualora li si vogliano riconfigurare tutti (cosa in genere non necessaria). Inoltre, dato che il sistema di *debconf* supporta un sistema di priorità delle domande che consente di evitare quelle al di sotto della priorità scelta, che si imposta appunto con `dpkg-reconfigure debconf`, si può usare l'opzione `-p` per forzare l'uso di una priorità diversa. Al solito per tutti i dettagli si consulti la pagina di manuale.

4.3 La gestione di utenti e gruppi

Tratteremo in questa sezione la gestione degli utenti e dei gruppi presenti nel sistema: vedremo i comandi utilizzati per crearli, eliminarli, e impostarne attributi e proprietà ed esamineremo le modalità con cui configurare le procedure di accesso al sistema, l'autenticazione e l'uso delle risorse assegnate agli utenti.

4.3.1 Una visione generale

Dato che GNU/Linux è un sistema multiutente abbiamo finora dato per scontato la presenza di utenti che potessero utilizzarlo. In realtà la questione non è affatto così immediata. Il kernel infatti supporta la presenza di utenti e gruppi associando a questi degli identificatori numerici che poi vengono usati nella gestione dei processi e dei file, ma come abbiamo abbondantemente ripetuto in sez. 1.1 tutta la gestione del sistema, compresa quella che permette il riconoscimento e l'accesso agli utenti, viene realizzata da appositi programmi.

Questo vale in particolare anche, come accennato in sez. 1.1.2, per quella procedura di collegamento al sistema che porta ad avere dei processi che vengono eseguiti dal kernel per conto di un certo utente. Perché questa sia effettuata però occorrono le *opportune informazioni* che permettano ai vari programmi che gestiscono la procedura (come `login`) di sapere quali utenti ci sono, come riconoscerli e quali risorse assegnargli.

Questo ci porta ad una delle caratteristiche fondamentali di un qualunque sistema multitutente: il concetto di *account*. Come avviene per una banca, per poter utilizzare il sistema si deve avere un "*conto*" presso di lui, che ci conceda l'accesso e l'uso delle risorse. Ovviamente

perché questo accada non solo occorrerà, come accennavamo in sez. 1.4.1, che un utente si identifichi appropriatamente, ma dovranno anche essere identificate le risorse che gli si mettono a disposizione.

Una delle parti più critiche dell'amministrazione del sistema è allora quella che permette di creare e mantenere le informazioni relative agli *account* degli utenti presenti, che dovranno contenere sia le informazioni necessarie all'identificazione degli stessi che quelle relative alle risorse messe loro a disposizione.

Tradizionalmente in un sistema unix-like l'autenticazione degli utenti viene fatta utilizzando un *username*, che è il nome che il sistema associa all'utente, ed una *password* segreta la cui conoscenza serve a dimostrare l'autenticità della propria identità. Come accennato in sez. 1.4.1 il sistema prevede anche la presenza di gruppi di utenti, che a loro volta vengono identificati da altri nomi (i *groupname*). Inoltre, come visto in sez. 1.2.2, ogni utente ha a disposizione una home directory per i propri file, e quando si collega al sistema gli viene messa a disposizione una shell, la *shell di login*, come illustrato in sez. 2.1.6. Tutte queste informazioni ed altre ancora sono quelle che vengono mantenute ed utilizzate dai programmi per la gestione di utenti e gruppi.

Infine si tenga presente che utenti e gruppi vengono utilizzati dal sistema non solo per gestire gli *account* delle persone fisiche che utilizzano il sistema, ma anche per associare una *identità* ad alcuni servizi. Ad esempio molti programmi server vengono eseguiti per conto di un utente a loro riservato, questo permette di mantenere separati i loro file rispetto a quelli di altri servizi e di non dare loro privilegi amministrativi,[15] aumentando la sicurezza del sistema.

Si ricordi infatti che dal punto di vista del kernel esistono solo gli identificatori numerici di utenti e gruppi illustrati in sez. 1.4.1 e che i corrispondenti nomi sono informazioni disponibili solo in user-space, proprio per la presenza di un sistema di gestione degli stessi, il *Name Service Switch*, che vedremo meglio in sez. 4.3.6. Per questo che un utente sia associato ad una persona fisica, o semplicemente utilizzato da un programma di servizio, per il kernel è del tutto indifferente, l'unica differenza che il kernel riconosce fra i vari utenti è quella fra l'amministratore e gli altri, e sta ai programmi di gestione degli utenti stabilire una politica che allochi opportunamente gli identificatori.

Nelle prime versioni di Unix tutte le informazioni relative agli utenti ed ai gruppi presenti nel sistema erano memorizzate su due file, `/etc/passwd` e `/etc/group`, che tratteremo meglio in sez. 4.3.2. Dato che questi, come tutti i file di configurazione del sistema, sono file di testo, in teoria non ci sarebbe nessuna necessità di programmi specifici per la loro gestione, visto che possono essere modificati a mano con un qualunque editor. Questa è una cosa che in certi casi è comunque utile saper fare, ad esempio per togliere una password di amministratore dal sistema contenuto in un disco montato a mano usando un sistema di recupero, poiché in quel caso i comandi andrebbero ad operare sulla configurazione del sistema di recupero, e non di quello che si vuole riparare.

Nei sistemi moderni però il meccanismo di gestione di utenti e gruppi è stato completamente modularizzato attraverso sia l'uso del *Name Service Switch* (vedi sez. 4.3.6) che di PAM (*Pluggable Authentication Modules*, vedi sez. 4.3.7). In questo modo è possibile mantenere le informazioni ed effettuare i relativi controlli usando i supporti più disparati (server NIS, vari database, server LDAP, ecc.), e quindi in generale non è più possibile andare ad effettuare le

[15] nella maggior parte dei casi questi, quando servono, sono garantiti solo all'avvio del servizio, ed eliminati non appena non sono più necessari.

modifiche a mano con un editor. L'uso di un supporto modulare però fa sì che in molti casi si possano utilizzare buona parte degli stessi comandi che in origine operavano solo sui file di testo in maniera trasparente rispetto al meccanismo con cui sono effettivamente gestite le informazioni.

4.3.2 Il database di utenti e gruppi

Nelle prime versioni di Unix tutte le informazioni relative ad utenti e gruppi venivano mantenute in due soli file: `/etc/passwd` e `/etc/group`. In seguito, con l'introduzione delle *shadow password*, ne sono stati aggiunti altri due: `/etc/shadow` e `/etc/gshadow`. Infine con l'uso di PAM (trattato in sez. 4.3.7) e del *Name Service Switch* (trattato in sez. 4.3.6) è divenuto possibile utilizzare i supporti più vari. A tutt'oggi però la modalità più comune per mantenere il database degli utenti e dei gruppi presenti su un sistema GNU/Linux resta quella di scrivere le relative informazioni sui file di testo appena citati, che sono quelli che tratteremo in questa sezione.

Fin dalle prime versioni di Unix la lista degli utenti viene mantenuta nel file `/etc/passwd`, chiamato così perché al suo interno venivano memorizzate anche le password. Il formato di `/etc/passwd` è molto semplice, per ogni utente deve essere presente una riga composta da 7 campi separati dal carattere "`:`" e non sono ammessi né commenti né righe vuote. Il significato dei sette campi è il seguente:

1. il nome dell'utente (l'*username*).
2. password cifrata (opzionale).
3. l'*user ID* (vedi sez. 1.4.1) dell'utente
4. il *group ID* del gruppo principale dell'utente.
5. nome e cognome dell'utente ed eventuali altri campi di commento separati da virgole; questo campo è detto anche "*Gecos*".[16]
6. home directory dell'utente (indicata con un pathname assoluto).
7. shell di login dell'utente.

Tutti i dettagli relativi al formato ed al significato dei vari campi di `/etc/passwd`, ed in particolare a quello del *Gecos*, si trovano sulla relativa pagina di manuale, accessibile con `man 5 passwd`. Un esempio del contenuto di questo file è il seguente:

/etc/passwd
```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:100:sync:/bin:/bin/sync
games:x:5:100:games:/usr/games:/bin/sh
man:x:6:100:man:/var/cache/man:/bin/sh
...
piccardi:x:1000:1000:Simone Piccardi,,,:/home/piccardi:/bin/bash
...
```

L'elenco dei gruppi viene invece mantenuto nel file `/etc/group`; il suo formato è analogo a quello di `/etc/passwd`: un gruppo per linea senza commenti e righe vuote. In però questo

[16] per maggiori informazioni sul significato di questa sigla si può consulti `http://en.wikipedia.org/wiki/Gecos`.

caso i campi presenti sono soltanto quattro, sempre separati dal carattere ":", ed indicano rispettivamente:

1. il nome del gruppo (il cosiddetto *groupname*).
2. la password del gruppo (si veda sez. 4.3.3).
3. il *group ID* (vedi sez. 1.4.1) del gruppo
4. la lista degli *username* degli utenti appartenenti al gruppo, separati da virgole.

Come per /etc/passwd si possono trovare tutti i dettagli sul formato ed il significato dei campi di /etc/group nella rispettiva pagina di manuale, accessibile con man group; un esempio del contenuto di questo file è il seguente:

/etc/group

```
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:
tty:x:5:piccardi,admin
...
piccardi:x:1000:
...
```

Si noti come in /etc/group non vengano riportati come membri di un gruppo gli utenti che hanno lo stesso come gruppo principale. Questa informazione infatti viene mantenuta in /etc/passwd, mentre qui vengono riportati solo gli utenti che hanno quel gruppo come gruppo ausiliario. Si noti inoltre come in entrambi gli esempi sia presente una "x" nel campo dedicato alla password; questo indica che nel caso sono attive le *shadow password*, su cui torneremo fra poco.

In genere per modificare i dati di questi file si possono utilizzare gli opportuni comandi di gestione, che vedremo in sez. 4.3.3; trattandosi di file di testo è comunque possibile effettuare interventi di emergenza anche con un semplice editor. Uno di questi potrebbe essere, qualora si fosse persa la password di root e non si riuscisse più ad entrare nel sistema, quello di far ripartire il computer con un CD di recupero o una distribuzione "*live*" e, montato il disco, togliere la "x" lasciando vuoto il campo della password. L'assenza di un campo per la password infatti fa sì che questa non venga più richiesta,[17] e al successivo riavvio si potrà così entrare nel sistema per ripristinarla senza doversi autenticare.

Ovviamente, dato che la creazione di utenti e gruppi è un compito riservato all'amministratore, entrambi i file devono appartenere a root ed essere protetti da scrittura. Essi però devono poter essere letti da chiunque, perché le informazioni mantenute in questi file sono quelle che vengono usate da tutti i programmi (ad esempio anche da ls e ps) per tradurre gli identificativi numerici del *group ID* e dell'*user ID* usati dal kernel nei corrispondenti *username* e *groupname*.

Il fatto che questi file siano leggibili da tutti può fare legittimamente sorgere il dubbio di come questo si possa conciliare con il fatto che vi vengano mantenute delle password che si suppone debbano restare segrete. Questo è possibile perché le password non sono mai memorizzate in

[17] questo è il comportamento tradizionale, oggi la cosa viene controllata dal sistema di PAM (vedi sez. 4.3.7) ed in genere la configurazione che viene adottata dalla maggior parte delle distribuzioni lo consente soltanto quando si è su una console autorizzata.

chiaro; il testo della password non viene scritto da nessuna parte nel sistema, quello che viene memorizzato è solo il risultato della cifratura della password, cioè il suo "*hash crittografico*" (si ricordi quanto detto in sez. 2.4.6).

Tutte le volte che si richiede una autenticazione quello che il sistema fa è semplicemente ricalcolare questo valore per la password che gli si fornisce e verificare che coincida con quello memorizzato. In realtà la procedura è leggermente più complessa perché per evitare che a password uguali corrispondano cifrature uguali ad esse viene aggiunto un valore casuale, detto *salt*, che serve a differenziarle in modo che due utenti non si possano accorgere di avere la stessa password. Questo *salt* viene memorizzato insieme alla cifratura e riutilizzato nella verifica.

Dato che solo se si è fornita la password originaria si potrà riottenere lo stesso risultato, in caso di coincidenza si è ottenuta l'autenticazione. Se il meccanismo crittografico utilizzato è valido, e lo si reputa tale quando qualunque metodo si possa usare per fare questa conversione inversa richiede una quantità di calcolo equivalente a quello di provare tutte le possibili password, si ha come conseguenza che risalire dal valore cifrato alla password originale è praticamente impossibile, per cui il rendere leggibile quest'ultimo non viene ritenuto un problema.

Benché l'algoritmo crittografico con cui si calcolano le password sia piuttosto robusto, mantenere leggibili le password, anche se cifrate, espone comunque ad un attacco a *forza bruta*, cioè alla possibilità che qualcuno tenti davvero di provarle tutte. L'algoritmo originale, il DES, infatti oggi non viene più usato proprio perché prevede delle chiavi di dimensione ridotta ed ha dei limiti sul numero di caratteri delle password (vengono usati solo i primi 8 caratteri) che rendono molto semplice un attacco a *forza bruta*. Per questo oggi si usano algoritmi alternativi come MD5 che prende password di dimensioni qualsiasi e utilizza chiavi più lunghe.

Ma se quando è stato creato Unix l'eventualità di un attacco a *forza bruta*, dati i computer dell'epoca, era impraticabile, con la potenza dei computer di oggi lo è molto meno. Questo anche perché esistono metodi per precalcolare grandi numeri di chiavi e compilare delle tabelle che consentono di ridurre notevolmente le richieste di calcolo a scapito di un maggior uso di memoria. Inoltre anche se oggi è diventato possibile usare algoritmi di crittografia che rendono più difficile un attacco di questo tipo, c'è sempre da fare i conti con la pigrizia degli utenti che tendono ad usare password come `pippo` o `ciccio` ed anche se è possibile forzare gli utenti ad usare password più solide (lo vedremo in sez. 4.3.7) ma non è detto che questo sia sempre possibile.

Per cui anche se gli attacchi a forza bruta non sono praticabili, è comunque piuttosto semplice, e ci sono programmi molto efficienti nel farlo, utilizzare quello che si chiama un *attacco a dizionario*, in cui invece che tutte le combinazioni possibili si provano solo quelle relative ad un dizionario di password probabili.[18] E non giovano neanche trucchetti come quello di scrivere le parole alla rovescia, aggiungere qualche cifra in cima o in fondo, invertire delle lettere o mettere un "`3`" al posto di una "`e`" o un "`1`" al posto di una "`i`" o un "`4`" al posto di una "`a`", perché sono tutti espedienti ampiamente noti, che qualunque programma decente di "*password cracking*" applicherà alle parole del suo dizionario.

Per questo motivo non è molto sicuro lasciare accessibili a chiunque le password, anche se nella loro forma cifrata, ed alcuni anni fa è stata effettuata una completa reimplementazione dei meccanismi di gestione degli utenti, con l'introduzione di quello che viene chiamato il sistema delle *shadow password*. Con questo nuovo sistema, oltre a spostare le password cifrate in un

[18] per una trattazione più accurata di queste problematiche si può consultare [SGL].

file a parte non accessibile in lettura, è stato anche possibile aggiungere una ulteriore serie di funzionalità di sicurezza.

Quando nel sistema sono abilitate le *shadow password* nel secondo campo di /etc/passwd ed /etc/group viene posta una "x", e le password cifrate vengono spostate in /etc/shadow ed /etc/gshadow. Nei primi tempi questa era una opzione di installazione, ma ormai sono molti anni che tutte le distribuzioni utilizzano le *shadow password* come default. Dato che in questo caso le altre informazioni necessarie ai programmi restano disponibili in /etc/passwd ed /etc/group si possono proteggere questi due nuovi file in lettura, così che solo l'amministratore possa accedervi.

Oltre alle password cifrate in /etc/shadow sono memorizzate una serie di ulteriori informazioni che permettono un controllo molto più dettagliato sulle modalità e le politiche di autenticazione di un utente, come il tempo di durata di una password, la data in cui la si è cambiata l'ultima volta, ecc. Il formato del file è analogo ai precedenti, e prevede 9 campi (sempre separati dal carattere ":") il cui significato è:

1. il nome dell'utente (*username*).
2. la password cifrata.
3. il giorno in cui la password è stata cambiata l'ultima volta (espresso in numero di giorni dal 1/1/1970).
4. il numero di giorni che devono passare dall'ultimo cambiamento prima che la password possa essere cambiata nuovamente.
5. il numero di giorni dall'ultimo cambiamento dopo i quali la password scade e deve essere necessariamente cambiata.
6. il numero dei giorni precedenti quello di scadenza della password in cui gli utenti vengono avvisati.
7. il numero dei giorni successivi a quello di scadenza della password dopo i quali l'utente viene disabilitato.
8. il giorno in cui l'utente viene disabilitato (espresso in numero di giorni dal 1/1/1970).
9. campo riservato, non viene utilizzato.

Al solito tutti i dettagli relativi al formato di /etc/shadow ed al significato dei vari campi possono essere trovati nella relativa pagina di manuale, un esempio del contenuto di questo file invece è il seguente:

/etc/shadow

```
root:n34MlzgKs8uTM:12290:0:99999:7:::
daemon:*:11189:0:99999:7:::
bin:*:11189:0:99999:7:::
sys:*:11189:0:99999:7:::
sync:*:11189:0:99999:7:::
games:*:11189:0:99999:7:::
...
piccardi:$1$KSRp2lZ3$s9/C2ms0Ke9UTaPpQ98cv1:11189:0:99999:7:::
...
```

Si noti come nell'esempio per alcuni utenti nel campo della password cifrata sia presente il carattere "*" (talvolta viene usato anche "!"). Questo è un modo, dato che una password cifrata non potrà mai avere questo valore, di impedire che il corrispondente utente possa autenticarsi

presso il sistema, e lo si usa normalmente quando si vuole disabilitare l'accesso ad un utente,[19] o per gli utenti dei servizi di sistema che non necessitano di eseguire un login.

Come accennato l'utilità del sistema delle *shadow password* è che questo consente anche di impostare delle politiche di gestione delle stesse. Ad esempio si può impostare un tempo di durata massima forzando gli utenti a non utilizzare sempre la stessa password, cosa che a lungo andare fa aumentare le possibilità che essa venga scoperta.

Dato che anche i gruppi possono avere delle loro password, anche in questo caso è stato previsto un secondo file non accessibile in lettura, `/etc/gshadow` in cui poterle mantenere, insieme ad altre informazioni. Il formato è analogo a quello degli altri file, i campi in questo caso sono 4 ed indicano rispettivamente:

1. nome del gruppo.
2. password cifrata.
3. utenti amministratori del gruppo (lista separata da virgole).
4. utenti appartenenti al gruppo (lista separata da virgole).

Di nuovo si è riportato a seguire un esempio del contenuto di `/etc/gshadow`, il cui formato, con la descrizione completa dei vari campi, è documentato nella relativa pagina di manuale:

/etc/gshadow

```
root:*::
daemon:*::
bin:*::
sys:*::
adm:*::
tty:*::piccardi,admin
...
piccardi:!::
...
```

4.3.3 I comandi per la gestione di utenti e gruppi

Benché sia possibile inserire a mano i dati nei vari file illustrati in sez. 4.3.2, in genere è opportuno utilizzare gli appositi comandi di gestione di utenti che si incaricano di effettuare le operazioni in maniera corretta. Il primo di questi comandi è `useradd`, che permette di aggiungere un nuovo utente al sistema (cioè di creare un *account*). Se usato senza opzioni il comando prende come argomento il nuovo username e si limita a creare un voce per il nuovo utente in `/etc/passwd` e `/etc/shadow`, inserendo dei valori di default in tutti i campi in cui ciò è possibile, ma non imposta la password, che resta disabilitata, e non crea la home directory.

In genere il comando assegna automaticamente al nuovo utente un *user ID*, utilizzando il primo numero libero maggiore di tutti gli altri *user ID* già assegnati agli altri utenti, partendo comunque da un valore minimo previsto dal sistema, che viene stabilito dalla propria distribuzione. Il valore minimo viene controllato dalla direttiva `UID_MIN` di `/etc/login.defs`, (vedi sez. 4.3.5), valori tipici sono 1000, usato da Debian e derivate, e 500, usato da RedHat e derivate;

[19] questo funziona soltanto se l'accesso prevede l'uso di una autenticazione basata sulle password, e non è vero se ci sono altri meccanismi di autenticazione, come ad esempio quello a chiavi crittografiche di SSH che tratteremo in sez. 8.3.3.

i valori fra 0 e detto limite sono usati normalmente per gli utenti corrispondenti ai servizi di sistema. Si può comunque impostare un valore numerico specifico indicandolo come parametro per l'opzione `-u` del comando.

Per quanto riguarda la home directory questa viene impostata automaticamente con il nome dell'utente aggiunto ad una base che di default è `/home` ma si può impostare una directory qualsiasi con l'opzione `-d`, che prende come parametro il relativo pathname assoluto. Inoltre con l'opzione `-m` si può richiedere anche la creazione della home directory, con tanto di creazione al suo interno di tutti i file contenuti nella directory `/etc/skel/` (vedi sez. 4.3.5).

In teoria si può impostare una password per l'utente direttamente con `useradd` usando l'opzione `-p`; questa però richiede come parametro una password specificata direttamente in forma cifrata, per cui normalmente non si usa mai questa opzione, ma si provvede ad eseguire il comando `passwd` in un secondo tempo.

Altre opzioni comuni sono `-s`, con cui si può indicare, passando come parametro il relativo pathname assoluto, una shell di login alternativa; `-g` con cui si specifica il gruppo principale, indicato per nome o per *group ID*, e `-G` con cui si impostano gli eventuali altri gruppi a cui l'utente appartiene, indicati con una lista dei rispettivi nomi separati da virgole.

Si tenga presente però che per poter eseguire le impostazioni manuali dei gruppi occorre che questi già esistano in `/etc/group`. Le versioni più vecchie del programma prevedevano, non specificando niente, l'assegnazione automatica ad un gruppo di default, indicato dalla variabile `GROUP` nel file `/etc/default/useradd`, predisposto dalla propria distribuzione (ad esempio su Debian e RedHat viene usato il generico gruppo `users` con *GID* 100). Le versioni più recenti invece si curano di creare automaticamente un gruppo con lo stesso nome scelto per l'utente, a meno che non si specifichi l'opzione `-N` che riporta al comportamento precedente.

Opzione	**Significato**
`-d`	imposta la home directory dell'utente.
`-u`	specifica un valore numerico per l'*user ID*.
`-p`	imposta la password, che deve essere passata come parametro, in forma cifrata.
`-s`	imposta la shell di default.
`-g`	imposta il gruppo primario dell'utente.
`-G`	imposta eventuali gruppi aggiuntivi.
`-o`	permette di specificare un *user ID* già esistente.
`-e`	imposta una data di scadenza per l'account.
`-f`	imposta dopo quanto tempo l'account viene disabilitato dopo la scadenza della password.
`-D`	mostra i default o li cambia se invocato con `-b`, `-s`, `-g`, `-e`, `-f`.
`-m`	copia il contenuto di `/etc/skel/` nella home.
`-b`	imposta la base da cui viene generato il valore di default della home directory.

Tabella 4.18: Principali opzioni del comando `useradd`.

Il comando inoltre consente, tramite l'opzione `-D`, di leggere o impostare i valori di default. In tal caso non si deve specificare nessun utente, e se usato senza opzioni il comando stamperà i valori correnti, mentre se utilizzato con una delle opzioni `-s`, `-g`, `-e`, `-f` cambierà il default per il rispettivo valore di impostazione (riscrivendo il contenuto di `/etc/default/useradd`).

Inoltre si può anche specificare l'opzione `-b` che indica la directory di base da cui viene creata la home directory di default, aggiungendo al valore passato come parametro (il valore preimpostato è `/home`) l'username dell'utente. Le principali opzioni sono riportate in tab. 4.18, comprese quelle che consentono di impostare alcuni attributi delle *shadow password*, l'elenco completo insieme a tutti i dettagli sul funzionamento del comando sono disponibili nella pagina di manuale accessibile con `man useradd`.

Una volta creato un utente è possibile modificarne le proprietà con il comando `usermod`. Il comando è simile a `useradd`, prende come argomento il nome dell'utente che si vuole modificare e molte opzioni (quelle elencate nella prima parte di tab. 4.18) sono identiche e hanno lo stesso significato. Si tenga presente però che l'uso dell'opzione `-G`, che permette di indicare i gruppi aggiuntivi di cui l'utente fa parte, richiede che si passi come parametro la lista completa (sempre separata da virgole) degli stessi. Questo significa che se non si inseriscono nella lista anche i gruppi di cui l'utente fa già parte, esso ne sarà rimosso. Ovviamente questo non è molto comodo, per cui le versioni più recenti del comando supportano l'uso dell'opzione `-a`, che consente di usare `-G` per aggiungere l'utente al gruppo passato come argomento mantenendo inalterati la presenza nei gruppi a cui esso già appartiene.

Infine `usermod` supporta le opzioni `-L` e `-U`, usate rispettivamente per bloccare e sbloccare l'accesso di un utente. Il blocco viene realizzato con l'apposizione del carattere "`!`" al campo contenente la password cifrata, così da far fallire comunque l'autenticazione, per lo sblocco il carattere viene semplicemente rimosso. Si sono riassunte le opzioni specifiche di `usermod` in tab. 4.19, al solito per i dettagli si consulti la pagina di manuale.

Opzione	**Significato**
`-l`	specifica un diverso username.
`-U`	sblocca l'account di un utente.
`-L`	blocca l'account di un utente.
`-a`	consente di usare `-G` per aggiungere un gruppo.

Tabella 4.19: Opzioni specifiche del comando `usermod`.

Per la gestione dei gruppi l'analogo di `useradd` è `groupadd` che permette di creare un nuovo gruppo, passato come argomento. Il comando permette di specificare un *group ID* specifico con l'opzione `-g`, altrimenti, come per `useradd`, viene preso il valore seguente al *group ID* più alto assegnato. Anche in questo caso, come per gli utenti, il *GID* viene sempre scelto all'interno dell'intervallo stabilito dalle direttive di `/etc/login.defs`, (l'argomento è trattato in dettaglio in sez. 4.3.5). Se il *group ID* richiesto con `-g` è già in uso il comando normalmente fallisce con un errore, se però si aggiunge l'opzione `-o` si può forzare l'uso di un gruppo duplicato, mentre con l'opzione `-f` si forza il successo del comando eseguendo la scelta di un *group ID* diverso da quello richiesto come avviene con il metodo ordinario.

Infine si può impostare un cosiddetto *gruppo di sistema* con l'opzione `-r`. Quella dei gruppi di sistema è comunque soltanto una convenzione, per il kernel un gruppo di sistema è un gruppo come tutti gli altri, è però invalso l'uso da parte di tutte le distribuzioni di usare valori bassi dei *group ID*, inferiori al limite iniziale usato per i gruppi degli utenti, da assegnare ai gruppi che vengono utilizzati per l'esecuzione di programmi che svolgono servizi nel sistema.

Analogamente a quanto avviene per gli utenti per modificare le proprietà di un gruppo si può usare il comando `groupmod`, in particolare il comando permette di cambiare il *group ID* usando

`-g`, supportando sempre l'opzione `-o` per poter specificare un gruppo già assegnato, e di cambiare il nome del gruppo con l'opzione `-n`. Al solito per l'elenco completo delle opzioni e tutti i dettagli si consultino le pagine di manuale.

I comandi `userdel` e `groupdel` permettono invece di cancellare rispettivamente un utente e un gruppo, da specificare come argomento. Nel caso di `groupdel` il comando non ha nessuna opzione specifica, ma non effettuerà la rimozione di un gruppo se questo risulta essere il gruppo primario di un utente, in tal caso occorre prima rimuovere l'utente con `userdel`. Si tenga conto che `userdel` si limita a cancellare l'utente se si vuole anche rimuoverne la home directory si dovrà usare l'opzione `-r`, ma si tenga presente che questo non assicura la cancellazione di tutti i file di proprietà dell'utente che potrebbero essere in altre directory diverse dalla sua home. Inoltre il comando prima di operare si assicura che l'utente che si vuole eliminare non sia collegato alla macchina rifiutandosi di proseguire altrimenti,[20] la cancellazione può però essere forzata con l'opzione `-f`.

Come alternativa ai precedenti, su Debian e derivate per la creazione e rimozione di utenti e gruppi sono presenti i comandi `adduser`, `addgroup`, `deluser` e `delgroup` che forniscono una interfaccia più semplice.[21] Con `adduser` ad esempio nella creazione dell'utente vengono richiesti pure i dati aggiuntivi e la password dal terminale, inoltre si può aggiungere un utente ad un gruppo semplicemente con il comando `adduser utente gruppo`. Nel caso della rimozione è inoltre possibile richiedere, tramite le opportune opzioni, anche la cancellazione di tutti i file dell'utente presenti nel sistema, e non solo quelli della sua home; al solito per i dettagli sul funzionamento si consultino le varie pagine di manuale ad essi dedicate.

Oltre ai comandi visti finora, che operano a livello generico su utenti o gruppi, ci sono una serie di comandi che permettono di modificare alcuni attributi specifici. Il più importante è `passwd`, che permette di impostare la password di un utente e tutte le proprietà ad essa connesse. L'utilizzo più comune è quello in cui si invoca il comando per cambiare la propria password, nel qual caso non è necessario nessun argomento; il comando chiederà la password corrente e poi la nuova password per due volte (la seconda per conferma, onde evitare errori di battitura).

Se si specifica un argomento questo indica l'utente di cui si vuole cambiare la password, ma si tenga presente che solo l'amministratore può cambiare la password di un altro utente, gli utenti normali possono cambiare solo la propria e soltanto dopo essersi autenticati fornendo la vecchia. Questo non vale per l'amministratore, che può cambiare la propria password senza dover fornire la attuale.

Oltre alla semplice operazione di cambiamento delle password `passwd` supporta molte altre funzionalità relative alla gestione delle stesse, ed in particolare per l'impostazione delle varie proprietà associate al sistema delle *shadow password* (trattate in sez. 4.3.2). Si sono riportate in tab. 4.20 le opzioni più rilevanti insieme al rispettivo significato, si tenga presente che esse possono essere utilizzate solo dall'amministratore. Infine il comando permette di bloccare e sbloccare l'uso di un account rispettivamente con le opzioni `-l` e `-u`, mentre l'opzione `-d` consente di cancellare totalmente la password, in tal caso però chiunque potrà entrare nel sistema conoscendo soltanto l'username.

Per gestire le informazioni di scadenza delle password di un utente è comunque disponibile un comando apposito, `chage`. Questo, a parte quando invocato con l'opzione `-l` che mostra i valori

---

[20] per far questo viene controllato quanto riportato in `/var/log/wtmp`, si può eseguire una verifica invocando il comando `last` (vedi sez. 2.4.4).

[21] il comando `adduser` è presente, ma solo come sinonimo per `useradd`, anche su RedHat e derivate.

Opzione	Significato
`-x`	imposta in numero massimo di giorni per cui la password rimane valida, passati i quali l'utente sarà forzato a cambiarla.
`-n`	imposta il numero minimo di giorni dopo il quale una password può essere cambiata, l'utente non potrà modificarla prima che siano passati.
`-w`	imposta il numero di giorni per i quali l'utente viene avvertito prima della scadenza della password.
`-i`	imposta il numero massimo di giorni per cui viene accettato il login dopo che la password è scaduta, passati i quali l'account sarà disabilitato.
`-e`	fa scadere immediatamente una password, forzando così l'utente a cambiarla al login successivo.

Tabella 4.20: Opzioni del comando `passwd` per la gestione delle informazioni relative alle *shadow password*.

attuali delle impostazioni, può essere usato solo dall'amministratore e prende come argomento l'username dell'utente su cui si vuole operare. Le opzioni del comando, che consentono di cambiare le varie proprietà di scadenza delle password, sono riportate in tab. 4.21; per maggiori dettagli si consulti al solito la pagina di manuale del comando.

Opzione	Significato
`-d`	imposta il giorno di ultimo cambiamento delle password, in numero di giorni dal primo Gennaio 1970, o nel formato YYYY-MM-DD (anno-mese-giorno).
`-E`	imposta il giorno di scadenza dell'account, oltre il quale non sarà più utilizzabile, usa lo stesso formato di `-d`, il valore -1 elimina la data di scadenza.
`-I`	imposta per quanti giorni dopo la scadenza della password l'account resta accessibile prima di essere bloccato.
`-m`	imposta la durata minima (in giorni) della password.
`-M`	imposta la durata massima (in giorni) della password.
`-W`	imposta il numero di giorni per i quali fornisce il preavviso della prossima scadenza della password.

Tabella 4.21: Opzioni del comando `chage` per la gestione delle informazioni di scadenza delle password.

Oltre alle proprietà associate alle password ci sono altri due comandi che consentono di modificare le altre caratteristiche degli utenti. Il primo di questi è `chfn` che permette di cambiare le informazioni mantenute nel campo chiamato *Gecos* (vedi sez. 4.3.2), in cui si scrivono il nome reale e altri dati relativi all'utente. Un utente può modificare solo i suoi dati, e solo quelli permessi dal parametro `CHFN_RESTRICT` di `login.defs` (vedi sez. 4.3.5), previa autenticazione, mentre l'amministratore può modificare tutti i dati di un utente qualunque. Se non si specifica nulla il comando presenta sul terminale i dati attuali richiedendo l'immissione di un nuovo valore, si può però specificare quest'ultimo direttamente a riga di comando come argomento per la rispettiva opzione. Per i dettagli e le singole opzioni si consulti al solito la pagina di manuale.

Il secondo comando è `chsh`, che permette ad un utente di cambiare la propria shell di login, impostandone una diversa come argomento per l'opzione `-s`, se non si specifica nulla la richiesta viene fatta in modalità interattiva sul terminale. Anche in questo caso un utente normale può operare solo su se stesso invocando il comando senza argomenti, e gli verrà comunque richiesta

l'autenticazione. L'amministratore invece può operare su qualunque utente senza restrizioni, passando il relativo username come argomento. Inoltre un utente normale non può indicare come shell un programma qualunque, ma soltanto una shell fra quelle elencate nel file `/etc/shells` (vedi sez. 4.3.5).

Abbiamo accennato in sez. 4.3.2 come, benché sia una caratteristica sostanzialmente inutilizzata e pressoché sconosciuta, è possibile assegnare una password anche ad un gruppo, inoltre con l'introduzione delle *shadow password* è possibile anche definirne degli amministratori, cioè dei membri del gruppo a cui è possibile delegare la capacità di aggiungere e rimuovere utenti dallo stesso. Il programma che permette di gestire tutto questo, ed in generale tutte le caratteristiche dei gruppi, è `gpasswd`.

Normalmente, come si può verificare nell'estratto riportato in sez. 4.3.2, i gruppi hanno una password disabilitata, vale a dire che il secondo campo di `/etc/gshadow` è impostato ad un valore ("*" o "`!`") che non corrisponde a nessun hash valido. Questo significa che possono far parte del gruppo solo gli utenti presenti come membri dello stesso in `/etc/group`,[22] se si imposta una password diventa invece possibile ad altri utenti diventare membri del gruppo, fornendo la relativa password.

La password di un gruppo si può impostare con `gpasswd` passando come argomento il nome del gruppo stesso. Il comando inoltre può essere usato per aggiungere o rimuovere utenti da un gruppo rispettivamente con le opzioni `-a` e `-d` (cui passare come parametro il nome dell'utente). Infine è possibile anche rimuovere completamente la password di un gruppo (consentendo a chiunque di diventarne parte) con l'opzione `-r`, mentre con l'opzione `-R` si disabilita l'uso della password, riportandosi alla situazione ordinaria.

Tutte queste operazioni possono essere compiute solo dall'amministratore di sistema o dagli amministratori del gruppo; questi possono essere impostati solo dall'amministratore di sistema tramite l'uso dell'opzione `-A` cui passare una lista di utenti separati da virgole. Sempre all'amministratore di sistema è riservato l'uso dell'opzione `-M` che consente di definire i membri ordinari del gruppo, passati sempre come lista di utenti separata da virgole.

Uno dei problemi che ci si trova ad affrontare con `passwd` e `gpasswd` è che entrambi richiedono che l'immissione della password avvenga da un terminale e non l'accettano qualora lo *standard input* sia un file o provenga da una redirezione di un comando precedente. Pertanto non sono di uso immediato qualora li si voglia usare in uno script, ad esempio per eseguire un cambiamento di password su un gran numero di utenti in maniera automatica.

In teoria in casi come questo si potrebbe pensare di usare `usermod` e `groupmod`, che consentono di impostare la password con l'opzione `-p` ma a parte le considerazioni di sicurezza, dato che il valore passato comparirebbe nella riga di comando, che è sempre visualizzabile nell'uscita di `ps`, una tale opzione richiede che la password sia passata in forma cifrata nel formato adeguato.

Per risolvere questo problema si possono usare due comandi, `chpasswd` e `chgpasswd`, creati appositamente per cambiare password rispettivamente a liste di utenti o di gruppi. Entrambi leggono la lista delle modifiche da fare dallo *standard input* che in questo caso non è necessario sia un terminale. Il formato della lista è analogo e prevede una riga per utente (o gruppo) divisa in due campi separati dal carattere "`:`", in cui il primo campo indica l'*username* (o il *groupname*) ed il secondo la password, in chiaro, che verrà cifrata dal comando stesso.

[22] in realtà vengono usati gli stessi dati presenti `/etc/gshadow`, che ha la precedenza in caso di differenze.

Si tenga presente che entrambi i comandi proseguono l'esecuzione in caso di fallimento per il cambiamento della password di un utente (o di un gruppo) passando a quello successivo e riportando un errore solo alla fine dell'esecuzione. Entrambi i comandi supportano l'opzione `-c` per indicare l'algoritmo crittografico da utilizzare (uno fra DES, MD5, NONE, e SHA256 o SHA512) e `-e` per indicare che le password vengono fornite già cifrate sullo *standard input*. Per i dettagli si consultino al solito le pagine di manuale.

4.3.4 Impersonare i ruoli di altri utenti e gruppi

Come già accennato in sez. 1.4.1 l'architettura di un sistema unix-like prevede fin dalla sue origini che tutti i programmi vengano eseguiti facendo sempre riferimento ad un utente ed ai gruppi di cui questo fa parte, ed anche come l'accesso al sistema sia vincolato al qualificarsi presso lo stesso assumendo una certa identità, per conto della quale si potranno poi effettuare tutte le operazioni successive.

Ci sono però dei casi in cui, pur essendosi collegati al sistema con un certo utente, diventa necessario assumere l'identità di un altro utente per eseguire altri compiti: il caso più classico in cui questo avviene è quello in cui si devono compiere operazioni sul sistema come amministratore. Ovviamente è sempre possibile scollegarsi dal sistema e ricollegarsi come amministratore, ma, specie se si è all'interno di una sessione grafica, questo non solo è scomodo ma anche pericoloso, dato che non è il caso di fare girare programmi ad interfaccia grafica con i privilegi di *root*.

Una regola d'oro per evitare guai infatti è quella di non usare mai i privilegi di amministratore se non quando strettamente necessario per le operazioni che si devono compiere. Per l'uso ordinario del sistema *root* non serve, ed usarlo per qualunque altro compito che non sia l'amministrazione di sistema significa correre dei rischi inutili anche solo per gli effetti che potrebbe avere un comando sbagliato.

Ovviamente se si è collegati si può sempre selezionare un'altra console e rieffettuare la procedura di login per assumere l'identità dell'utente che ci interessa, se si hanno le relative credenziali di accesso, ma per semplificare la procedura ci vengono incontro una serie di comandi grazie ai quali diventa possibile eseguire programmi per conto di altri utenti o altri gruppi direttamente dalla propria sessione di lavoro. Dato che un utente normale non può eseguire programmi per conto di altri, per far questo i suddetti comandi sono tutti dotati di *suid bit* per l'utente di amministrazione, che è l'unico che ha questa capacità, e richiedono, a meno di non essere lanciati dall'amministratore stesso, che chi li esegue fornisca delle opportune credenziali di accesso (in genere una password).

Il comando classico usato per cambiare identità è `su`, che in genere viene invocato senza argomenti per diventare amministratore, il nome sta appunto per ***s**uper **u**ser*. Si può comunque diventare un utente qualsiasi passando al comando il relativo *username*. Il comando richiede di autenticarsi con la password dell'utente richiesto, a meno che non si sia già l'amministratore nel qual caso non viene richiesta nessuna password. Questo consente all'amministratore di *impersonare* qualunque utente senza doverne conoscere la password.

Se non si specifica altro il comando dopo di che esegue una shell per conto dell'utente che si è scelto di impersonare, ed è anche previsto, come ausilio per la verifica delle operazioni, che ogni operazione eseguita con `su` venga opportunamente registrata nel *syslog*, nella *facility* `auth` (vedi sez. 3.2.3).

Di default come effetto del comando viene lanciata la shell di login specificata dal sesto campo della voce di `/etc/passwd` associata all'utente che si va ad impersonare, da cui poi si potranno eseguire i comandi per conto di questi. Si può però eseguire anche un singolo comando (sempre all'interno di detta shell) specificandolo come parametro per l'opzione `-c`.

Si può inoltre richiedere l'uso di una shell diversa rispetto a quella di login, specificandola come parametro per l'opzione `-s`. Questo però è possibile soltanto se all'utente è stata assegnata una shell elencata in `/etc/shell`; in caso contrario, se l'utente ha quella che viene chiamata (vedi sez. 4.3.5) una *restricted shell*, a meno che il comando non sia stato invocato dall'amministratore, l'opzione viene ignorata.

Infine si tenga presente che anche se `su` consente di avere una shell eseguita per conto di un altro utente la directory di lavoro e l'ambiente restano quelli della shell da cui si era partiti; se ci si vuole trovare in una situazione analoga a quella che avrebbe l'utente scelto dopo il suo login, si può usare l'opzione `-l` (o anche semplicemente `-`). Si sono riportate le principali opzioni del comando in tab. 4.22, al solito per i dettagli e l'elenco completo si faccia riferimento alla pagina di manuale.

Opzione	Significato
`-c` *cmd*	richiede l'esecuzione del solo comando passato come argomento.
`-s` *shell*	richiede l'uso della shell passata come argomento.
`-l` o `-`	richiede la creazione di un ambiente analogo a quello risultante dopo il login dell'utente richiesto.

Tabella 4.22: Principali opzioni del comando `su`.

In sez. 4.3.2 abbiamo detto che impostando una password su un gruppo si può permettere ad altri utenti che non ne sono membri di farne parte; non abbiamo però spiegato come farlo. La modalità più comune è attraverso l'uso del comando `newgrp`. Il comando permette infatti di eseguire una nuova shell aggiungendo il gruppo passato come argomento ai propri gruppi ausiliari.

Alternativamente si può usare anche il comando `sg`, che però, in analogia con `su`, supporta l'opzione `-c` che consente di eseguire un singolo comando con i privilegi del nuovo gruppo. Entrambi i comandi prendono come argomento il nome del gruppo del quale si vogliono avere i diritti e richiedono la relativa password, ma la password non è richiesta se i comandi vengono eseguiti dall'amministratore. Al solito per i dettagli si consultino le rispettive pagine di manuale.

Benché `su` venga utilizzato comunemente per ottenere temporaneamente i privilegi di amministratore (o di un altro utente) una volta che questi sono stati ottenuti non c'è più nessuna limitazione. Esistono invece molte situazioni in cui può tornare utile far eseguire ad un normale utente un programma per conto dell'amministratore o di un altro utente, ma si vuole che questi possa eseguire in questo modo solo quello specifico programma.

Per soddisfare questa esigenza esiste un apposito comando, `sudo`, che consente di delegare l'esecuzione per conto di altri di singoli comandi, script o programmi. Il programma è controllato da un apposito file di configurazione in cui si può specificare quale programma può essere lanciato per conto di quale utente e quali sono gli utenti cui questo è consentito.

Per utilizzare `sudo` basta specificare come argomento il comando che si vuole eseguire insieme alle eventuali opzioni. Se questo è consentito dalla configurazione verrà richiesta la *propria*

password, a meno che il comando venga eseguito dall'amministratore o che si chieda un'esecuzione per conto del proprio utente, dopo di che il comando verrà eseguito. Se non si specifica altro il comando viene eseguito per conto dell'amministratore, ma si può richiedere l'esecuzione per conto di un qualunque altro utente specificando prima del comando l'opzione `-u` seguita o dall'username o dall'UID.

Si tenga conto che una volta che ci si sia autenticati per l'esecuzione di un comando `sudo` mantiene l'autorizzazione valida per un breve intervallo di tempo durante il quale non è necessario fornire nuovamente la password. Il default è di 15 minuti, ma si può impostare un valore diverso con l'opzione `timestamp_timeout` in `/etc/sudoers`. In questo modo si possono far eseguire più comandi di seguito fornendo la password una sola volta, semplificando una eventuale sessione di lavoro. Il comando registra comunque, sia presso il *syslog* sulla *facility* `auth` che su un eventuale file specificato nella configurazione, tutte le volte che ne viene richiesta l'esecuzione, che questa abbia successo o meno.

Un'altra funzionalità di `sudo`, che si attiva usando l'opzione `-e` o invocando il comando come `sudoedit`, è quella che consente di delegare la modifica di un singolo file. In tal caso infatti delegare ad un utente l'uso di un editor con i privilegi di amministratore non è una soluzione, in quanto così si consentirebbe di modificare un file qualunque.

Con questa opzione invece viene prima creata una copia del file passato come argomento, sulla quale viene invocato l'editor di default dell'utente, quello indicato dalla variabile di ambiente `EDITOR`, o quello indicato in `/etc/sudoers`, ed una volta eseguite le modifiche sulla copia queste vengono riportate sul file originale. Le altre principali opzioni sono riportate in tab. 4.23, per l'elenco completo si consulti al solito la pagina di manuale.

Opzione	**Significato**
`-e`	invece di lanciare un comando consente la modifica del file passato come argomento, è equivalente all'invocazione del comando come `sudoedit`.
`-u` *`user`*	richiede l'esecuzione per conto dell'utente `user` (specificato per *username* o UID).
`-v`	rinnova il tempo di validità della sessione all'interno del quale non viene richiesta una password.
`-S`	richiede la lettura della password dallo *standard input* invece che dal terminale.

Tabella 4.23: Principali opzioni del comando `sudo`.

Il grande vantaggio di `sudo` è che il comando consente di realizzare in userspace, con una enorme flessibilità, un meccanismo di delega per specifici compiti amministrativi a singoli utenti o gruppi, senza dover ricorrere a meccanismi di controllo di accesso più sofisticati a livello di kernel, tanto che una distribuzione come Ubuntu ha addirittura disabilitato l'uso di *root*, usando `sudo` per assegnare all'utente che si crea in fase di installazione la possibilità di eseguire qualunque comando con privilegi amministrativi.

La potenza e la flessibilità di `sudo` hanno però un costo, che è quello di una adeguata configurazione del programma stesso, che avviene tramite il file `/etc/sudoers`. Purtroppo in questo caso la documentazione del file stesso, disponibile nella relativa pagina di manuale, è come minimo poco utilizzabile, gli autori infatti la hanno espressa in forma di Backus-Naur estesa (EBNF)[23]

[23] la EBNF (*Extended Backus-Naur Form*) è un modo di esprimere una sintassi generica in una forma compatta

il che ha permesso a loro di scriverla in maniera compatta e precisa, scaricando però lo sforzo necessario per tradurla in un significato più comprensibile sul povero utente finale, costretto ad impararsi la notazione EBNF anche quando ne avrebbe fatto volentieri a meno.

Per questo motivo cercheremo di dare qui una descrizione imprecisa, ma sperabilmente più intellegibile, del formato del file, che resta comunque molto complesso. La parte principale del file è quella delle regole che consentono di specificare quale utente può eseguire quale comando per conto di quale altro utente e su quali macchine.

Si possono infatti assegnare regole che si applicano solo a certe macchine all'interno di una rete, o a certe reti o a singole macchine. Tutto questo quando si ha a che fare con una singola macchina non ha alcun significato, ma potrebbe essere utile per usare lo stesso file di configurazione su una serie di macchine o all'interno di una rete. In realtà questa scelta lascia molto perplessi, visto che la complessità aggiuntiva nella sintassi di configurazione non pare fornire vantaggi significativi rispetto al mantenere diverse versioni del file sulle singole macchine.

Oltre a questo è inoltre possibile definire anche degli *alias* per ciascuno dei citati componenti delle regole, per raggruppare più utenti (o comandi, o nomi di macchine) sotto una opportuna etichetta. Si possono infine impostare dei default per le modalità di funzionamento del comando, anche questi specificabili per gruppi di utenti, macchine o comandi.

Al solito le righe che iniziano per "`#`" vengono considerate commenti, e le righe vuote vengono ignorate, inoltre si possono scrivere regole o *alias* su più righe terminando ciascuna con il carattere "`\`" per indicare di proseguire sulla seguente. La sintassi di base di una regola è nella forma:

```
utente ALL = (altroutente) /path/comando, /path/comando
```

Una regola viene espressa come una serie di componenti; il primo componente, specificato all'inizio della riga, identifica l'utente che è abilitato ad usare `sudo` per eseguire i comandi previsti nella stessa; questo primo componente può essere un singolo username, come nel precedente esempio, o il nome di un gruppo preceduto dal carattere "`%`", che consentirà l'uso della regola a tutti i membri del gruppo stesso, o un *alias*, con il quale si possono specificare liste di utenti, gruppi e altri *alias*. Si può altresì usare la parola chiave `ALL`, che significa qualunque utente.

Il secondo componente, separato dal primo da uno o più spazi, nel precedente esempio è specificato con la parola chiave `ALL`, il cui significato è ovvio. Questo componente servirebbe ad indicare la macchina o l'elenco di macchine, o la rete, ecc. su cui la regola è attiva, ma siccome faremo sempre riferimento ad una configurazione che vogliamo sia effettiva sulla macchina su cui la installiamo useremo sempre questo valore con cui si garantisce un accesso valido su qualunque macchina. Se si hanno esigenze diverse su macchine diverse si potranno sempre usare dei file `/etc/sudoers` contenenti regole diverse, o studiarsi la sintassi completa delle modalità con cui si indicano i nomi delle macchine.

Il terzo componente, separato dal secondo dal carattere uguale, nel caso dell'esempio riporta la specificazione, messa fra parentesi tonde, dell'utente per conto del quale si potranno eseguire i comandi indicati nelle componenti successive. Questo componente è opzionale, e può essere omesso, nel qual caso l'esecuzione verrà fatta per conto dell'amministratore. L'utente può essere indicato sia per username, come nell'esempio, che per UID (con l'indicazione del relativo valore

---

e precisa; il problema è appunto che essendo una forma compatta, è anche poco chiara, e lascia all'utilizzatore il tutt'altro che banale compito di *decomprimere* l'informazione in essa contenuta; il risultato è che ci vuole un lavoro notevole anche solo per capire le cose più elementari.

numerico preceduto dal carattere "#"), che con il nome di un gruppo (anche qui preceduto dal carattere "%") nel qual caso si potranno usare i comandi per conto degli utenti membri del detto gruppo ed infine anche con un *alias* o con la parola chiave ALL che consente l'esecuzione per conto di qualunque utente.

Al terzo componente segue, separato da spazio, il comando di cui si vuole consentire l'esecuzione; se ne può anche specificare più di uno, separandoli con delle virgole. Anche qui vale la parola chiave ALL per indicare un comando qualsiasi, inoltre si può usare il valore riservato sudoedit che serve ad indicare che si sta abilitando la modifica di un file (da indicare come argomento) e non l'esecuzione di un comando.

I comandi devono essere specificati con un pathname assoluto al relativo programma, ma si possono usare le *wildcard* del *filename globbing* per fare selezioni multiple. Se non si specifica altro essi potranno essere usati con qualunque argomento, si possono però specificare anche opzioni ed argomenti e sudo consentirà l'esecuzione del comando soltanto con quelle opzioni e quegli argomenti, ma in tal caso si tenga presente che i caratteri usati nella specificazione delle regole, come ":", "=", ",", "\" dovranno essere adeguatamente protetti con un "\" se presenti nelle opzioni o negli argomenti del comando stesso. Infine se non si vuole che vengano usati argomenti si dovrà far seguire al comando la stringa vuota espressa con "".

La sintassi delle regole viene ulteriormente complicata dal fatto che è possibile esprimere più blocchi dei componenti successivi al primo separandoli con il carattere ":", così da avere una regola che indica comandi diversi eseguiti per conto di utenti o su macchine diverse; ma di nuovo in casi come questi è più semplice e chiaro scrivere due regole separate piuttosto che usare una sintassi complessa e poco leggibile.

Tag	**Significato**
NOPASSWD	evita che sudo richieda l'autenticazione dell'utente che intende eseguire il comando.
PASSWD	rimuove l'effetto di un precedente NOPASSWD.
SETENV	vengono mantenute nell'esecuzione le variabili di ambiente presenti nella shell del chiamante, evitando le eventuali restrizioni imposte nella configurazione di default.
NOSETENV	rimuove l'effetto di un precedente SETENV.
EXEC	evita che un comando possa lanciare al suo interno altri programmi (ad esempio che dall'interno di vi si possa eseguire una shell).
NOEXEC	rimuove l'effetto di un precedente EXEC.

Tabella 4.24: I tag per le indicazioni dei comandi in /etc/sudoers.

Inoltre nello specificare un comando sono disponibili alcuni *tag* che consentono di controllarne le modalità di esecuzione, questi vengono specificati indicandoli prima del comando a cui si applicano seguiti dal carattere ":"; una volta specificati valgono anche per i comandi successivi nella regola, a meno che l'indicazione non venga soprascritta da un'altra specificazione. L'elenco dei *tag* ed il relativo significato è illustrato in tab. 4.24.

Come accennato una delle funzionalità presenti in /etc/sudoers è quella che consente la creazione di *alias* che permettono di esprimere in maniera più sintetica gruppi di utenti, o di macchine o liste di comandi; gli *alias* sono espressi nella forma di assegnazione di una stringa identificativa al rispettivo valore, preceduta da una delle direttive di definizione riportate in tab. 4.25.

Direttiva	**Significato**
`User_Alias`	Un *alias* utilizzabile per indicare gli utenti che possono eseguire un certo comando, prende come argomenti *username*, *groupname* (preceduti da "`%`") o altri alias.
`Host_Alias`	Un *alias* utilizzabile per indicare una lista di macchine o reti.
`Runas_Alias`	Un *alias* utilizzabile per indicare gli utenti per dei quali si può eseguire un certo comando, prende come argomenti *username*, UID (preceduti da "`#`"), *groupname* (preceduti da "`%`") o altri alias.
`Cmd_Alias`	Un *alias* utilizzabile per indicare una lista di comandi.

Tabella 4.25: Le direttive per la creazione di alias in `/etc/sudoers`.

Infine tutta una serie di proprietà generali concernenti le modalità di funzionamento di `sudo`, come il tempo per il quale non viene richiesta nuovamente una password dopo essersi autenticati, possono essere impostate tramite la direttiva `Defaults`. Questa prende come argomenti una serie di nomi di opzioni che identificano le suddette proprietà da configurare, separate da virgole. La sintassi prevede che l'impostazione venga effettuata in forma di assegnazione di un valore al nome dell'opzione, ad eccezione del caso di opzioni che hanno valore logico, nel qual caso basta citare l'opzione per attivarla o farla precedere da un "`!`" per disattivarla. A complicare l'impostazione si aggiunge il fatto che si possono definire delle impostazioni di default per singoli gruppi di utenti, macchine e comandi, in una forma che non tratteremo, si consulti nel caso la pagina di manuale, facendo riferimento agli esempi.

Opzione	**Significato**
`env_reset`	valore logico; di default a `on`, richiede la ripulitura dell'ambiente lasciando definite soltanto le variabili `LOGNAME`, `SHELL`, `USER`, `USERNAME` e le eventuali `SUDO_*`.
`env_keep`	stringa; nella forma di lista di nomi separati da virgole, imposta l'elenco delle variabili di ambiente che vengono preservate oltre a quelle di `env_reset`.
`setenv`	valore logico; di default a `off`, richiede che tutte le variabili di ambiente vengano preservate.
`passwd_tries`	valore numerico; imposta il numero di tentativi di immissione delle password.
`timestamp_timeout`	valore numerico; imposta il tempo, in minuti, per cui `sudo` ritiene la sessione valida e non richiede la password.
`editor`	stringa; nella forma di lista di nomi separati da virgole, imposta la lista di editor che possono essere usati con `visudo`.
`syslog`	imposta la *facility* del *syslog* su cui mandare i messaggi, il default è `authpriv`.

Tabella 4.26: Le principali opzioni per i default di `/etc/sudoers`.

Come citato le opzioni possono avere valori numerico, testuale o logico, a seconda del tipo di funzionalità che controllano; in generale non è necessario specificare nessuna opzione, in quando i default hanno valori ragionevoli. Una delle opzioni più importanti è l'opzione logica `env_reset`, che richiede la ripulitura dell'ambiente passato al comando.[24] Questo viene fatto per

[24]su Debian è l'unica opzione che viene impostata esplicitamente nell'`/etc/sudoers` installato di default, su RedHat oltre a questa viene usata anche `env_keep`, le versioni recenti del comando comunque tengono attiva

evitare possibili abusi che potrebbero avvenire sfruttando le variabili di ambiente: si potrebbe ad esempio definire `LD_LIBRARY_PATH` (vedi sez. 3.1.2) per fare usare ad un programma delle librerie *opportunamente* modificate, per far eseguire ad un programma delle operazioni non troppo benevole.

Si può controllare con maggiore dettaglio l'ambiente passato al comando utilizzando anche l'opzione `env_keep`, che prende come valore la lista, separata da virgole, delle variabili di ambiente che possono essere mantenute nell'esecuzione del comando. Se invece si vuole preservare completamente l'ambiente si può usare l'opzione `setenv`. In tab. 4.26 si sono riportate le opzioni principali, al solito per l'elenco completo si consulti la pagina di manuale.

Come esempio di configurazione che illustra la sintassi del file riportiamo il contenuto di un `/etc/sudoers` che consente l'accesso amministrativo agli utenti del gruppo `admin` (secondo l'approccio usato da Ubuntu), mentre consente all'utente `webmaster` di riavviare i servizi di Apache con l'uso del comando `apachectl`:

/etc/sudoers

```
# /etc/sudoers
#
# This file MUST be edited with the 'visudo' command as root.
#
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset

User_Alias   WEBMASTER = webmaster
Cmnd_Alias   APACHECTL = /usr/sbin/apachectl

# User privilege specification
root         ALL = (ALL) ALL
%admin       ALL = (ALL) ALL
WEBMASTER    ALL = NOPASSWD: APACHECTL
```

dove si sono usati (anche se nel caso sono del tutto superflui) due alias per definire l'utente ed il comando da far eseguire per il controllo del demone di Apache.

In questo file sono presenti tre regole, che illustrano abbastanza bene le sintassi più comuni. Anzitutto si noti che per tutte quante viene usata la indicazione generica "`ALL =`" che rende la regola valida su qualunque macchina. La prima si applica ad un singolo utente (nel caso l'amministratore) e la seconda ai membri del gruppo `admin` ma i permessi sono identici, con `(ALL)` si consente di impersonare qualunque utente, e con il successivo `ALL` di usare qualunque comando. La terza regola usa invece un *alias* per indicare l'utente precedentemente definito a cui consente l'uso del comando `apachectl` (anch'esso definito in un *alias*) con privilegi di amministratore (si noti che non c'è una indicazione dell'utente da usare per l'esecuzione del programma), inoltre avendo usato il *tag* `NOPASSWD` non verrà neanche richiesta una password.

Infine si tenga presente che benché sia possibile eseguire la modifica direttamente, la modalità più corretta per effettuare una riconfigurazione di `sudo` è quella di usare il comando `visudo` che consente di chiamare un editor per modificare `/etc/sudoers` in maniera sicura. Il programma infatti evita che si possano effettuare modifiche simultanee (bloccando l'accesso al file se è in

questa opzione.

corso l'uso da parte di qualcun altro) e prima di salvare la nuova versione esegue un controllo di base, per evitare errori.

4.3.5 Le configurazioni della gestione degli utenti e degli accessi

In sez. 4.3.2 abbiamo trattato i file in cui classicamente vengono mantenuti i dati di utenti e gruppi, mentre in sez. 4.3.3 abbiamo visto i comandi che consentono di gestirli; ci restano da trattare tutti quei file di configurazione che controllano da una parte le modalità con cui si creano utenti e gruppi e dall'altra le modalità con cui questi accedono al sistema.

Agli albori di Unix l'unica procedura per collegarsi al sistema era quella del login da terminale che, come accennato in sez. 2.1.2, è gestita dai due programmi `getty` e `login`. Le modalità con cui questi programmi (ed in particolare il secondo) garantiscono l'accesso vengono impostate attraverso una serie di file di configurazione, che sono quelli che tratteremo in questa sezione. In realtà al giorno d'oggi tutte le distribuzioni usano i *Pluggable Authentication Modules* (PAM, che tratteremo in sez. 4.3.7), e buona parte dei file di configurazione che un tempo venivano usati direttamente da `login` adesso vengono utilizzati attraverso in modo generico con questa libreria, che ne consente l'uso a qualunque programma abbia necessità di gestire l'accesso al sistema.

Il primo file che prendiamo in considerazione è `/etc/issue`, che contiene il messaggio di presenza stampato sui terminali prima dell'accesso dell'utente. Questo file viene usato da `getty` per essere stampato sul terminale prima della stringa "`login: `" dopo la quale viene richiesto l'username; è possibile modificarlo in modo da personalizzare la schermata di accesso al sistema.

Il contenuto del file viene stampato integralmente, ma si possono inserire, attraverso l'uso del carattere "`\`", delle direttive che permettono di stampare alcune informazioni dinamiche; ad esempio con "`\s`" si inserisce automaticamente il nome del sistema operativo, con "`\l`" il nome del terminale, con "`\n`" il nome della macchina. I principali valori sono riportati in tab. 4.27, mentre l'elenco completo è riportato nella pagina di manuale di `getty`.

Carattere	Significato
`\d`	data completa.
`\l`	nome del terminale.
`\m`	architettura (i486, PPC, ecc.).
`\n`	nome della macchina (*hostname*).
`\r`	versione del kernel.
`\s`	nome del sistema (Debian, RedHat, ecc).
`\t`	ora del giorno.

Tabella 4.27: Principali caratteri di estensione per `/etc/issue`.

Analogo a `/etc/issue` è `/etc/issue.net`, che viene usato al suo posto quando il collegamento viene effettuato via rete con `telnet`; si tenga conto che questo vale solo per un login remoto eseguito con `telnet`, se invece si usa `ssh` (vedi sez. 8.3) detti file normalmente vengono ignorati.

Un secondo file utilizzato all'ingresso del sistema, questa volta quando il login è stato completato, è `/etc/motd` che contiene il messaggio di benvenuto mostrato a tutti gli utenti che si sono collegati con successo. Il nome del file sta per *message of the day*, e scrivere un messaggio in questo file è una modalità veloce per mandare un avviso a tutti gli utenti che entrano nel sistema dall'interfaccia a riga di comando. Al contrario del precedente questo file viene gestito da PAM con il modulo `pam_motd.so`, per cui può essere utilizzato non solo sul collegamento da

terminale, ma anche da tutti i programmi che consentono l'accesso al sistema, come `ssh` (vedi sez. 8.3).

Un altro file di controllo della procedura di *login* è `/etc/securetty`. Questo file contiene la lista delle *console* da cui si può collegare l'amministratore di sistema (cioè l'utente *root*). Viene usato dal programma `login`, anzi per essere precisi dal modulo di PAM `pam_securetty.so` (vedi sez. 4.3.7), che legge da esso i nomi dei dispositivi di terminale dai quali è consentito l'accesso. Un esempio di questo file (preso da una Debian Squeeze) è il seguente:

/etc/securetty
```
console
vc/1
vc/2
...
tty1
tty2
...
```

Il formato del file prevede che ogni linea definisca un nome di dispositivo di un terminale sul quale è possibile il login, le linee vuote o che iniziano per "#" vengono ignorate. La pagina di manuale fornisce una descrizione completa. Dato che i terminali virtuali associati alle connessioni di rete usualmente non sono indicati in questo file, normalmente non è possibile utilizzare un programma come `telnet` (vedi sez. 7.7.1) per collegarsi da remoto come amministratore. Benché sia possibile consentirlo aggiungendo una riga al contenuto appena illustrato, questo non è assolutamente una buona idea per cui se proprio volete farlo dovrete studiarvelo da soli.[25]

Quando si vuole bloccare temporaneamente l'accesso al sistema degli utenti ordinari, l'amministratore può creare il file `/etc/nologin`. Questo non è propriamente un file di configurazione, dato che il suo contenuto è ininfluente, la sua presenza serve solo a dire alla procedura di collegamento al sistema che solo l'amministratore può entrare. Dato che anche l'effetto di questo file viene gestito con PAM, ed in particolare con il modulo `pam_nologin.so` (vedi sez. 4.3.7), l'accesso può essere bloccato in generale, anche dall'interfaccia grafica. La sua rimozione ripristinerà l'accesso per tutti gli utenti.

Oltre a quelli appena citati esistono vari altri file che consentono di controllare le modalità della procedura per l'accesso al sistema ed in particolare controllare alcune risorse assegnate agli utenti, oggi tutto questo viene gestito attraverso PAM per cui rimandiamo la trattazione di questi aspetti più avanzati alla sez. 4.3.7.

Tratteremo però qui `/etc/login.defs`, che un tempo era il principale file deputato alla configurazione della procedura di *login*, e conteneva anche le impostazioni di `login` per controllare le modalità di autenticazione ed accesso, come il numero di possibili errori nel fornire la password, o il tempo per cui si attende la sua immissione.

Come accennato nelle distribuzioni recenti tutto quello che riguarda l'autenticazione dell'utente e la gestione della sessione di lavoro viene gestito tramite PAM, per cui `login` non fa più alcun uso di questo file, anche se spesso vi si trovano riferimenti a direttive obsolete ormai non più utilizzate. Esso però resta come file di configurazione dell'intero sistema di gestione degli utenti, ed in particolare viene utilizzato per la configurazione di parametri usati dai programmi di gestione trattati in sez. 4.3.3.

[25] in generale non è comunque una buona idea neanche quella di usare `telnet`, quindi è meglio se proprio lasciate perdere completamente la cosa.

Parametro	Significato
`CHFN_RESTRICT`	indica quali campi del *Gecos* possono essere cambiati con `chfn` da un utente ordinario, indicati con una lettera identificativa, per i dettagli si consulti la pagina di manuale.
`GID_MAX`	imposta il valore massimo del *group ID* assegnabile coi comandi `useradd` e `groupadd`.
`GID_MIN`	imposta il valore minimo del *group ID* assegnabile coi comandi `useradd` e `groupadd`.
`PASS_MAX_DAYS`	imposta il valore di default per la massima durata della validità di una password, usato al momento della creazione dell'utente.
`PASS_MIN_DAYS`	imposta il valore di default per la minima durata della validità di una password, usato al momento della creazione dell'utente.
`PASS_WARN_AGE`	imposta il valore di default per il numero di giorni precedenti la scadenza di una password per i quali viene mandato un avviso, usato al momento della creazione dell'utente.
`UID_MAX`	imposta il valore massimo dell'*user ID* assegnabile col comando `useradd`.
`UID_MIN`	imposta il valore minimo dell'*user ID* assegnabile col comando `useradd`.
`MAIL_DIR`	indica la directory della posta dell'utente, viene usato dai comandi di gestione dell'utente per creare, spostare o rimuovere la posta dello stesso.
`USERDEL_CMD`	se definito indica il comando che verrà eseguito alla rimozione di un utente (ad esempio per compiere controlli aggiuntivi come la rimozione di eventuali stampe sospese).

Tabella 4.28: Principali parametri di configurazione di `/etc/login.defs`.

Il formato di `login.defs` è molto semplice, al solito sono considerati commenti le righe che iniziano per "`#`" ed ignorate le righe vuote. Ciascuna riga contiene l'assegnazione di un parametro di configurazione, fatta nella forma di un nome che lo identifica seguito dal valore da assegnargli, separato da uno o più spazi. A seconda del parametro il valore può essere una stringa, un numero, o un valore logico (specificato nella forma `yes` o `no`).

Abbiamo già accennato ad alcuni di questi parametri, come `UID_MIN` e `UID_MAX` in sez. 4.3.3, un elenco di quelli più rilevanti è stato riportato in tab. 4.28, l'elenco completo si trova nella pagina di manuale associata al file. Un esempio del contenuto di questo file, estratto della versione installata su una CentOS 5.2, è il seguente:

/etc/login.defs

```
MAIL_DIR       /var/spool/mail
PASS_MAX_DAYS       99999
PASS_MIN_DAYS  0
PASS_MIN_LEN   5
PASS_WARN_AGE  7
UID_MIN              500
UID_MAX              60000
GID_MIN              500
GID_MAX              60000
...
```

Si tenga presente infine che se si fa riferimento al contenuto effettivo di `/etc/login.defs` vi si potranno trovar definiti anche una gran quantità di parametri non citati nella pagina di manuale; alcuni di questi sono semplicemente obsoleti e non hanno nessun effetto in quanto gran parte delle impostazioni presenti un tempo, ed in particolare quelle usate da programmi come `login`,

`passwd` e `su`, viene oggi effettuata attraverso i moduli di PAM. Alcuni di questi però, anche a seconda di come sono stati costruiti i comandi citati, potrebbero essere tenuti in conto come valori di riserva qualora le impostazioni eseguite con PAM non funzionassero.

Un altro file utilizzato dal sistema di gestione degli utenti è `/etc/shells`, che contiene la lista delle shell valide che l'utente può selezionare con il comando `chsh` (vedi sez. 4.3.3). Il file utilizza il solito formato in cui le righe inizianti per "`#`" sono considerate commenti e le righe vuote sono ignorate. Ogni riga non vuota contiene un solo campo che specifica il pathname completo del programma che può essere usato come shell. Un utente che voglia cambiare la propria shell di default potrà usare solo una shell fra quelle che sono indicate in questo file. In questo modo l'amministratore può lasciare all'utente la libertà di modificare la propria shell di login, restringendola però alle shell autorizzate.

Il file però ha degli effetti anche sul funzionamento di altri programmi, in particolare viene usato per verificare se un utente è un utente ordinario con accesso al sistema, sulla base della presenza della sua shell di login, che si ricordi è quella definita nell'ultimo campo di `/etc/passwd` (vedi sez. 4.3.2), in questo file.

Si dice infatti che un utente è dotato di una *restricted shell* quando la sua shell di login non è fra quelle elencate in `/etc/shells`.[26] Sono esempi tipici di utenti associati ad una *restricted shell* quelli utilizzati per i servizi di sistema,come gli utenti associati ai servizi di fornitura di pagine web o gestione della posta elettronica, che spesso hanno come shell programmi come `/bin/false` (vedi sez. 2.1.5) o `/usr/bin/nologin` che non consentono comunque di collegarsi al sistema. Quest'ultimo è un semplice programma che stampa un messaggio dicendo che l'utente non è utilizzabile e ritorna un codice di errore; `nologin` è stato creato apposta per essere usato come *restricted shell* da assegnare agli utenti di servizio che non necessitano di avere un accesso al sistema.

La presenza di una *restricted shell* in `/etc/passwd` viene usata da vari programmi per modificare il proprio comportamento. Ad esempio alcuni server FTP rifiutano l'accesso ad username corrispondenti ad utenti che non hanno una shell valida fra quelle elencate in `/etc/shells`, mentre, come accennato in sez. 4.3.4, `su` si rifiuta di eseguire un comando per conto di un altro utente usando una shell diversa da quella di login se questi ha una *restricted shell*.

Si tenga conto comunque che qualora l'utente abbia comunque un accesso ad una shell l'impedirgli di cambiare la shell di login al di fuori di quelle di `/etc/shells` non significa che gli si può impedire di usare un'altra shell, questi infatti potrà comunque installare nella propria home directory un'altra shell ed usare quella al posto della shell di login, lanciandola semplicemente come un qualunque altro programma; non potrà però cambiare quella con cui entra nel sistema al login.

Infine, come ultimo argomento relativo alle configurazioni che attengono alla gestione degli utenti, dobbiamo trattare la directory `/etc/skel/`. Come accennato in sez. 4.3.3, nella creazione di un nuovo utente è possibile richiedere la creazione della sua home directory; questa può anche essere creata con uno *scheletro* di contenuti, che siano gli stessi per ogni nuovo utente. Con questo si può far sì ad esempio che alcune impostazioni personali di programmi di uso comune[27] siano predefinite ed assegnate in modo uniforme alla creazione di ogni nuovo utente.

[26] si rammenti che questo è un concetto diverso rispetto alla modalità ristretta della `bash` vista in sez. 2.1.6, anche se in inglese anche questa viene chiamata allo stesso modo.

[27] si ricordi che, come accennato in sez. 3.1.1, molti programmi consentono all'utente di usare configurazioni personalizzate con dei file nella propria home.

La directory `/etc/skel/`, come il nome stesso suggerisce, è quella che contiene questo scheletro. Tutti i file e le directory che si vuole siano creati nella home directory dei nuovi utenti possono essere messi in questa directory. Tutti i file presenti in `/etc/skel/` saranno automaticamente copiati nella relativa home alla creazione di ogni nuovo utente, i permessi dei file verranno mantenuti, mentre la proprietà degli stessi verrà assegnata al nuovo utente ed al suo gruppo principale.

Il contenuto tipico di questa directory sono gli scheletri dei file di personalizzazione della shell degli utenti: ad esempio nel caso di Debian Squeeze il contenuto di default prevede ad esempio i file `.bashrc`, `.profile`, `.bash_logout`, ma è possibile spingersi oltre ed inserire al suo interno anche altre configurazioni, ad esempio CentOS vi crea anche lo scheletro delle directory `.kde` e `.mozilla`.

4.3.6 Il *Name Service Switch*

Una delle funzionalità di base fornite fin dai primi Unix dalla libreria standard del sistema (nel caso di Linux la *glibc*) è quella che consente di associare agli identificativi numerici utilizzati dal kernel dei nomi simbolici di più immediata fruizione. In generale tutto ciò viene realizzato grazie ad una serie di funzioni che permettono ai programmi di ottenere queste informazioni, come le corrispondenze fra nomi degli utenti e *user ID* o fra nomi dei gruppi ed il *group ID*, che come abbiamo visto in sez. 4.3.2, venivano mantenute nei file `/etc/passwd` e `/etc/group`.

In generale oltre che per i nomi degli utenti e dei gruppi, il problema di ottenere delle corrispondenze fra un identificativo numerico ed un valore simbolico si presenta anche in altri casi, ad esempio con le corrispondenze fra nomi assegnati alle macchine e gli indirizzi di rete.[28] Tradizionalmente queste corrispondenze, e le ulteriori informazioni ad esse collegate, venivano mantenute in opportuni file di configurazione posti sotto `/etc`.

Con l'evolversi delle funzionalità disponibili sui sistemi più moderni si è progressivamente manifestata la necessità di mantenere questo tipo di informazioni anche in maniera diversa, come quella di centralizzarle per interi gruppi di macchine tramite opportuni servizi di rete. Gli esempi più comuni sono il NIS (*Network Information Service*) ed LDAP (*Lightweight Directory Access Protocol*) due servizi di rete con cui si possono centralizzare le informazioni relative agli utenti.

Con l'introduzione di estensioni che consentissero l'uso di queste nuove funzionalità si presentava però anche il problema di come indicare alle varie funzioni di libreria dove prendere le informazioni. All'inizio questo veniva fatto introducendo tutta la casistica possibile nell'implementazione delle funzioni stesse, con degli ovvi problemi di estensibilità e compatibilità. Per risolvere il problema allora è stata creata una interfaccia generica, il *Name Service Switch*, che permette di demandare a delle librerie esterne, configurabili in maniera indipendente, le modalità con cui queste informazioni vengono ottenute. Il sistema venne introdotto la prima volta nella libreria standard di Solaris, la libreria standard GNU ha ripreso in seguito lo stesso schema.

Il grande vantaggio del *Name Service Switch* è che diventa possibile definire in maniera modulare ed estendibile sia delle classi di informazioni, cosicché qualora si debba fornire qualche nuovo servizio si ha l'infrastruttura già pronta, che il supporto su cui queste informazioni sono

[28] torneremo su questo argomento in sez. 7.4.2 quando tratteremo questo tipo di corrispondenze associate ai parametri relativi alla rete.

mantenute, che può essere su file, database o servizi di rete; è possibile inoltre indicare al sistema anche in quale ordine utilizzare le varie fonti, permettendo anche configurazioni ibride.

Le modalità di funzionamento del *Name Service Switch* vengono gestite attraverso il suo file di configurazione principale che è `/etc/nsswitch.conf`. Un esempio di questo file, come installato su una Debian Squeeze, è il seguente:

/etc/nsswitch.conf

```
# /etc/nsswitch.conf
#
# Example configuration of GNU Name Service Switch functionality.
# If you have the 'glibc-doc' and 'info' packages installed, try:
# info libc "Name Service Switch" for information about this file.

passwd:         compat
group:          compat
shadow:         compat

hosts:          files dns
networks:       files

protocols:      db files
services:       db files
ethers:         db files
rpc:            db files

netgroup:       nis
```

Il formato del file è sempre lo stesso, le linee vuote o che iniziano per "`#`" vengono ignorate, le altre linee indicano una opzione. La pagina di manuale contiene una lista completa delle opzioni disponibili. La prima colonna, terminata dal carattere "`:`" indica una classe di informazioni, di seguito vengono elencate, separate da spazi, le modalità con cui queste informazioni possono essere ottenute; la ricerca dei dati sulle varie fonti sarà eseguita nell'ordine in cui queste sono indicate. L'elenco delle classi di informazioni disponibili è riportato in tab. 4.29, ulteriori dettagli possono essere ottenuti consultando la pagina di manuale del file o la sezione *Name Service Switch* delle pagine *info* delle *glibc*.

Delle varie classi di informazioni riportate in tab. 4.29 le prime tre sono utilizzate per gestire in maniera modulare i dati relativi ad utenti e gruppi, esattamente come questi sono presenti negli omonimi file sotto `/etc` visti in sez. 4.3.2. Le altre attengono ad altre proprietà, principalmente legate agli identificativi usati dai protocolli di rete, che tratteremo più avanti in sez. 7.4.

Per ciascuna modalità di gestione delle informazioni deve esistere una opportuna libreria che garantisca l'accesso alle stesse con quella modalità. Queste librerie si trovano tutte sotto `/lib/` e devono avere un nome del tipo `libnss_`***`NOME`***`.so.`*`X`*, dove *`X`* fa riferimento alla versione delle *glibc* che si sta usando e vale 1 per le *glibc* 2.0 e 2 per le *glibc* 2.1 e successive, mentre ***`NOME`*** indica il tipo di supporto per quell'informazione, così come viene scritto in `/etc/nsswitch.conf` (nel caso dei supporti citati nell'esempio sarà `files`, `db`, `compat`, ecc.).

Di norma non c'è niente da cambiare in `/etc/nsswitch.conf`, a meno che non si debba aggiungere un ulteriore supporto da cui ottenere le informazioni. Ad esempio nel caso delle informazioni relative agli utenti si potrebbe voler utilizzare un sistema di autenticazione centralizzato su un servizio di rete come LDAP. In quel caso si dovrà installare l'apposito pacchetto di supporto, ed

Classe	Tipo di corrispondenza
shadow	corrispondenze fra username, *shadow password* ed altre informazioni relative alla gestione delle password (vedi sez. 4.3.2).
passwd	corrispondenze fra username, identificatori di utente e gruppo principale, shell di default, ecc. (vedi sez. 4.3.2).
group	corrispondenze fra nome del gruppo e proprietà dello stesso (vedi sez. 4.3.2).
aliases	alias per la posta elettronica.
ethers	corrispondenze fra indirizzo IP e *MAC address* della scheda di rete (vedi sez. 7.6.4).
hosts	corrispondenze fra nome a dominio e indirizzo IP (vedi sez. 7.4.3).
netgroup	corrispondenze gruppo di rete e macchine che lo compongono.
networks	corrispondenze fra nome di una rete e suo indirizzo IP (vedi sez. 7.4.4).
protocols	corrispondenze fra nome di un protocollo e relativo numero identificativo (vedi sez. 7.4.4).
rpc	corrispondenze fra nome di un servizio RPC e relativo numero identificativo (vedi sez. 7.6.5).
services	corrispondenze fra nome di un servizio e numero di porta (vedi sez. 7.4.4).

Tabella 4.29: Le diverse classi di corrispondenze definite all'interno del *Name Service Switch*.

inserire il relativo nome (nel caso `ldap`) in posizione opportuna nell'elenco dei supporti su cui sono disponibili le informazioni.[29]

La configurazione di default del *Name Service Switch*, quella vista nell'esempio precedente, prevede soltanto l'uso dei file classici trattati in sez. 4.3.2, ma questo, pur restando ancora oggi quello più diffuso, è solo uno dei metodi per mantenere le informazioni riguardo gli *account* degli utenti. Pertanto esaminare il contenuto di questi file non è detto sia sufficiente ad identificare tutti gli utenti di un sistema, in quanto in generale le informazioni potrebbero essere mantenute anche altrove.

Per questo motivo è disponibile un apposito comando, `getent`, che consente di interrogare in maniera generica il sistema del *Name Service Switch* ed ottenere una lista completa anche quando le informazioni sono distribuite su più supporti. Il comando richiede come argomento il nome della classe di informazioni gestite dal *Name Service Switch* per la quale si vogliono ottenere i dati, da indicare con uno dei valori riportati in tab. 4.29, dopo di che esegue una ricerca su tutti i supporti attivi per tale classe e stampa a video il risultato.

Come si può notare da tab. 4.29 le classi relative alle informazioni degli utenti e dei gruppi gestite dal *Name Service Switch* hanno un nome corrispondente al file in cui sono mantenute dal sistema delle *shadow password*, ed anche il formato con cui esse vengono mostrate attraverso l'uso di `getent` segue la stessa regola. Pertanto volendo ottenere una lista completa degli utenti si potrà eseguire il seguente comando, ottenendo un risultato analogo al contenuto di `/etc/passwd`:

```
piccardi@hain:~/truedoc/corso$ getent passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
...
```

[29] per un esempio dettagliato di questa configurazione si consulti la sez. 2.1.2 di [IS-LDAP].

```
piccardi:x:1002:1002:Simone Piccardi,,,:/home/piccardi:/bin/bash
admin:x:1003:1003:Utente Amministrativo,,,:/home/admin:/bin/bash
```

dove però, se presenti, compariranno anche i dati mantenuti in tutti gli eventuali altri supporti utilizzati dal *Name Service Switch*.

4.3.7 I *Pluggable Authentication Modules*

La necessità di una maggiore flessibilità della gestione degli utenti non riguarda soltanto la possibilità di mantenere su diversi supporti i dati loro associati, ma anche e soprattutto quella di permettere la gestione di diverse modalità per gestire l'autenticazione. A questo provvede il sistema chiamato *Pluggable Authentication Modules*, in breve PAM, introdotto inizialmente su Solaris e poi portato anche su GNU/Linux.

L'idea è quella di realizzare un meccanismo che ricopra per i programmi di autenticazione il ruolo che ha il *Name Service Switch* per quelli che richiedono una corrispondenza fra identificatori numerici e simbolici. Dato che in questo caso la richiesta è più sofisticata il sistema sarà più complesso, ma in sostanza si tratta sempre della definizione di una interfaccia di programmazione generica per cui tutti i programmi che necessitano di compiere una qualche forma di autenticazione (come `login` o `passwd`) lo fanno attraverso una libreria che fa da intermediario rispetto al meccanismo con cui poi le operazioni vengono effettivamente svolte, così che questo possa essere, in maniera totalmente trasparente ai programmi scelti, sostituito con un altro.

Nel gestire l'insieme dei metodi per l'autenticazione e l'accesso al sistema, PAM suddivide il lavoro in quattro tipologie di servizio indipendenti fra loro, illustrate in tab. 4.30, dove si sono riportate anche le quattro parole chiave usate nelle configurazioni per identificarle.

Servizio	**Significato**
`account`	fornisce il servizio di verifica della presenza di un *account* e della relativa validità (le password sono scadute, si può accedere al servizio, ecc.).
`auth`	fornisce il servizio di verifica dell'identità di un utente, in genere sulla base di un meccanismo di sfida/risposta (come dare la password), ma può essere esteso in maniera generica (permettendo l'uso anche di dispositivi ausiliari come smart-card, apparati biometrici ed altro).
`password`	fornisce il servizio di aggiornamento dei meccanismi di autenticazione (come cambiare la password), ed è strettamente collegato ad `auth`.
`session`	fornisce i servizi di preparazione per le risorse che devono essere messe a disposizione dell'utente (ad esempio il montaggio della home directory) all'inizio di una sessione di lavoro e quelli relativi alla liberazione delle stesse alla fine della sessione.

Tabella 4.30: Le quattro tipologie di servizi forniti da PAM.

La flessibilità di PAM è dovuta al fatto che il sistema è implementato come una libreria dinamica che permette di caricare una serie di *moduli*, usualmente mantenuti in `/lib/security`, che realizzano diverse funzionalità in una o più delle classi di tab. 4.30. Una delle caratteristiche più interessanti di PAM è che questi moduli possono essere *impilati* in modo da combinare le funzionalità di ciascuno nella fornitura di un determinato servizio. Inoltre il sistema permette di utilizzare impostazioni diverse per le differenti applicazioni che lo usano.

La configurazione di PAM può essere effettuata in due modi; quello più comune è l'uso di una serie di file, uno per ciascuna applicazione che usa il sistema, posti nella directory `/etc/pam.d/`.

In questo modo infatti è molto più semplice per le singole applicazioni mantenere un proprio file di configurazione. Ogni applicazione che necessita dell'uso del sistema sarà controllata da un suo file il cui nome corrisponde a quello del relativo servizio, in genere questo coincide con quello dell'applicazione stessa, si avranno cioè file come `login`, `su`, `passwd`, ecc. Uno di questi nomi, `other`, è riservato e serve per indicare le regole di default che verranno applicate tutte le volte che non viene richiesto un servizio specifico.

Ciascun file contiene una regola per riga, usata per invocare un determinato modulo; si può proseguire una regola su più righe terminandole con il carattere "\", ed al solito righe vuote ed inizianti per "#" vengono ignorate. Una regola è sempre composta da almeno tre campi separati da spazi (o tabulatori), al terzo di questi campi, che indica il modulo di PAM da utilizzare, possono seguire ulteriori campi che specificano degli argomenti per il modulo, in numero variabile che dipende dal modulo stesso. La forma generica di una regola da utilizzare in un file di `/etc/pam.d/` pertanto è:

```
type  control  module-path  [arguments ...]
```

Il primo campo, denominato `type`, indica, con una delle parole chiave di tab. 4.30, la classe di servizi a cui la regola fa riferimento. Il secondo campo, `control`, serve a definire il comportamento che si dovrà avere in caso di successo o fallimento del compito assegnato al modulo. Per indicare un valore in questo campo nella gran parte dei casi si può usare una sintassi semplificata che prevede l'utilizzo di una delle parole chiave riportate in tab. 4.31, tramite le quali si indica l'azione che deriva dal controllo effettuato.

Controllo	**Significato**
`requisite`	il fallimento del modulo invocato comporta l'immediata terminazione della procedura di valutazione, gli altri eventuali moduli non verranno invocati.
`required`	il fallimento del modulo comporta il fallimento della procedura di valutazione, ma gli altri moduli specificati nelle regole seguenti vengono comunque invocati.
`sufficient`	il successo del modulo comporta il successo dell'operazione, fintanto che nessun precedente modulo `required` è fallito.
`optional`	il successo o il fallimento del modulo non ha nessuna influenza sul risultato finale a meno che questo non sia l'unico modulo utilizzato.

Tabella 4.31: I principali valori per il campo di controllo dei risultati usati nei file di configurazione di PAM.

Infine il terzo campo indica il file (in genere una libreria condivisa, quindi un file `.so`) contenente il codice del modulo. Questo viene normalmente espresso con un pathname relativo alla directory `/lib/security/` dove vengono usualmente installati i moduli, ma si può comunque usare un file qualunque, specificando un pathname assoluto. Al nome del modulo può seguire una lista (separata da spazi) degli argomenti da passare a quest'ultimo, che varia da modulo a modulo a seconda delle funzionalità da esso fornite, anche se come vedremo esistono alcuni valori utilizzabili per qualunque modulo.

Una forma alternativa di configurazione, oggi totalmente in disuso, è quella dell'uso del file `/etc/pam.conf` in cui si inseriscono tutte le regole di gestione che starebbero dentro i file di `/etc/pam.d` ma se questa è presente `/etc/pam.conf` viene comunque ignorato. Il file prevede una

linea per ciascuna regola ed al solito si ignorano le righe vuote e quelle inizianti per "#". La differenza con il precedente formato è che queste prendono un ulteriore campo iniziale che indica il servizio cui la regola fa riferimento (quello che in /etc/pam.d sarebbe il nome del file), pertanto una regola sarà nella forma:

```
service  type  control  module-path  [arguments ...]
```

dove service indica il nome dell'applicazione cui si applica la regola (quello che con /etc/pam.d è il nome del file).

La configurazione di un programma per l'uso PAM prevede che nel corrispondente file sotto /etc/pam.d si elenchino, per ciascuna delle quattro classi di servizio illustrate in tab. 4.30, i moduli che si vuole siano da esso utilizzati. Come accennato questi possono essere *impilati*, vale a dire che si può richiedere per un singolo servizio l'uso di più moduli in successione in modo da poter eseguire diversi tipi di richieste o di operazioni di controllo e gestione. Ad esempio si può voler usare una procedura di autenticazione che prenda in considerazione sia le credenziali poste nei file di sez. 4.3.2 che quelle poste in un qualche sistema di centralizzazione delle informazioni come LDAP; in questo caso si specificheranno due moduli diversi, uno per autenticarsi sui file classici e l'altro per utilizzare LDAP.

Il sistema prevede che per ciascuna tipologia di servizio si elenchino i moduli che si vogliono utilizzare. Questi, normalmente, verranno eseguiti uno dopo l'altro nell'ordine in cui sono elencati. Sia l'ordine di esecuzione che il risultato finale dipendono però dall'azione di controllo richiesta in risposta al risultato di ciascun modulo, ad esempio con sufficient e requisite si interrompe immediatamente la scansione della *pila*, mentre azioni come optional e required fanno proseguire con l'esecuzione dei moduli successivi.

Un altro aspetto dell'esecuzione di ciascun modulo, anche questo dipendente dall'azione di controllo indicata, riguarda l'effetto del risultato ottenuto dal modulo stesso sul risultato finale restituito da PAM. Ad esempio il fallimento ottenuto nell'esecuzione di un modulo per cui si è utilizzato required o requisite comporta il fallimento dell'intera procedura, optional è ininfluente meno che questo non sia l'unico risultato ottenuto, mentre sufficient ne comporta il successo, ma solo se non c'è stato un precedente fallimento di un required.

Oltre alle azioni di controllo di tab. 4.31 si possono usare come valori per il campo control altre due parole chiave: la prima è include, che consente di indicare al posto di un modulo un altro file da cui includere le impostazioni, in questo modo si possono mettere configurazioni comuni a più servizi in un solo file. Delle configurazioni presenti nel file incluso verranno usate solo quelle che si applicano al tipo di servizio specificato prima di include, questo significa che occorrerà ripetere la direttiva per ciascuno dei servizi di tab. 4.30.[30]

La seconda parola chiave è substack, il cui effetto è analogo a quello di include con la differenza che le regole incluse dal file vengono considerate come parte di una pila secondaria. Questo comporta che quando le azioni di controllo impostate nel file incluso richiedono la terminazione della scansione questa avviene solo per la parte inclusa con tale direttiva e non per le altre regole contenute nel file da cui è usato substack.

[30]nel caso di Debian viene fornita una sintassi non standard, nella forma di una direttiva @include file da mettere su una sola riga, che consente di utilizzare tutte le configurazioni messe dentro file, vengono poi creati per le configurazioni di default anche quattro file common-*, uno per ciascuno dei servizi di tab. 4.30, che vengono inclusi da praticamente tutti i file di /etc/pam.d/.

Infine benché sia la scansione della pila dei moduli che il relativo risultato sia gestibile nella maggior parte dei casi con le azioni di controllo espresse nella forma semplificata di tab. 4.31, esistono casi in cui occorre un controllo più dettagliato, ottenibile soltanto attraverso la sintassi completa che consente di specificare la singola azione da associare a ciascun possibile valore di ritorno restituito dal modulo invocato. Per usare questa sintassi al posto delle parole chiave di tab. 4.31 si deve specificare una lista di assegnazioni nella forma `valore=azione` compresa fra parentesi quadre. Con `valore` si indica il nome simbolico associato al valore di ritorno, e con `azione` l'azione di controllo che si vuole gli sia associata.

Azione	Significato
`bad`	indica che il valore di ritorno ottenuto deve essere considerato come un fallimento della richiesta effettuata al modulo, e se questo è il primo fallimento avuto nella scansione della pila questo sarà usato anche come risultato finale.
`die`	equivalente all'uso di `bad` ma con l'ulteriore effetto di concludere immediatamente la scansione della pila.
`done`	equivalente all'uso di `ok`, ma con l'ulteriore effetto di concludere immediatamente la scansione della pila.
`ignore`	il valore di ritorno ottenuto viene ignorato e non avrà nessun effetto sul risultato finale.
`ok`	indica che il codice di ritorno deve essere considerato come un successo della richiesta effettuata al modulo; ma questo verrà utilizzato solo come risultato finale solo se non sono presenti fallimenti precedenti.
`reset`	cancella lo stato corrente ottenuto dai risultati dei precedenti moduli della pila, ripartendo da zero dal modulo successivo.

Tabella 4.32: Azioni associabili ad un valore di ritorno di un modulo di PAM.

L'interfaccia di PAM prevede infatti che tutti i moduli possono restituire una serie ben precisa di valori di ritorno, ciascuno dei quali ha un preciso significato. Alcuni dei più comuni sono `success`, dal significato evidente, `ignore`, che indica la richiesta di ignorare il modulo stesso (cui in genere si assegna l'azione omonima), `new_authtok_reqd`, che indica la richiesta di una ulteriore credenziale di autenticazione, `user_unknown` che indica che l'utente è inesistente.

La lista completa dei nomi associati ai possibili valori di ritorno è riportata nella pagina di manuale della configurazione, accessibile con `man pam.conf`, ma la documentazione del loro significato è praticamente nulla.[31] Inoltre si può utilizzare il nome speciale `default` per comprendere tutti i valori non esplicitamente indicati in risposta ai quali impostare una azione prestabilita.

Come azione associabile a ciascun valore di ritorno si può specificare o un intero positivo `n` o una delle parole chiave riportate in tab. 4.32. Nel primo caso si richiede di saltare all'interno della pila dei moduli per andare ad eseguire quello che segue il modulo corrente di `n` posizioni, senza che questi contribuisca al risultato finale; nel secondo caso l'azione sarà invece quella associata alla parola chiave secondo quanto illustrato in tab. 4.32. L'espressione dei codici semplificati di tab. 4.31 nei termini di questa sintassi completa è la seguente:

`required` equivale a `[success=ok new_authtok_reqd=ok ignore=ignore default=bad]`

[31] viene fatto riferimento al file `/usr/include/security/_pam_types.h` che nei commenti ha qualche breve spiegazione.

requisite equivale a [success=ok new_authtok_reqd=ok ignore=ignore default=die]

sufficient equivale a [success=done new_authtok_reqd=done default=ignore]

optional equivale a [success=ok new_authtok_reqd=ok default=ignore]

Una possibile configurazione che non rientra in questo schema semplificato potrebbe essere la seguente, che si è ripresa da quanto specificato per il modulo `pam_securetty.so` per la configurazione di `login` su una Debian Squeeze:

```
[success=ok ignore=ignore user_unknown=ignore default=die]
```

che è simile all'uso di `requisite`, ma che evita il fallimento immediato in caso di utente non esistente, facendo arrivare ai moduli successivi allo scopo di evitare che si possa distinguere dalla procedura di login se un utente esiste o meno.

La potenza di PAM sta nella grande varietà di moduli che mette a disposizione per realizzare i compiti più disparati. Ciascuno di questi prevede, come accennato, la possibilità di essere invocato passandogli degli argomenti che consentono di controllarne il comportamento. Prima di esaminare i moduli più importanti, ed il relativo funzionamento, è però opportuno citare il fatto che esistono una serie di argomenti generici, riportati in tab. 4.33, che possono essere utilizzati per qualunque modulo, anche se in realtà hanno effetto soltanto se il modulo stesso ne prevede l'utilizzo.

Argomento	Significato
`use_first_pass`	il modulo non deve richiedere una password all'utente ma usare quella già fornita in una precedente invocazione di un altro modulo; se questa non è valida all'utente non saranno fatte ulteriori richieste (l'argomento è valido solo per i servizi `auth` e `passwd`).
`try_first_pass`	il modulo deve usare la password già fornita in una precedente invocazione di un altro modulo, ed eseguire direttamente una richiesta qualora questa non sia valida (l'argomento è valido solo per i servizi `auth` e `passwd`).
`expose_account`	segnala al modulo che può essere più permissivo nel rivelare informazioni riguardo l'utente (in genere questo viene evitato per non fornire appigli ad un attaccante).
`use_authtok`	forza il modulo a non richiedere una nuova password (quando si esegue una cambiamento della stessa) ma ad usare quella proveniente da un modulo precedente (l'argomento è valido solo per il servizio `passwd`).

Tabella 4.33: Alcuni argomenti generici che possono essere passati ai moduli di PAM.

In particolare sono molto utili argomenti come `use_first_pass` e `try_first_pass` che consentono, quando si utilizzano più moduli che devono eseguire una operazione richiedendo una password, di non ripetere la richiesta della stessa ad ogni invocazione, facendo invece riferimento a quanto immesso nella invocazione di un modulo precedente.

Il primo dei moduli che prendiamo in considerazione è `pam_unix`,[32] che è senza dubbio uno dei più importanti in quanto è quello che fornisce autenticazione e gestione delle risorse degli utenti

[32] in questa trattazione indicheremo i moduli per nome, tralasciando il `.so` finale, che però deve essere usato quando li si richiamano nelle configurazioni di `/etc/pam.d/`.

secondo le modalità classiche di Unix ed è il default usato dalla gran parte delle distribuzioni. Il modulo utilizza i file di sez. 4.3.2 per quanto riguarda sia la verifica che il cambiamento della password, sia per i controlli ausiliari relativi alla validità dell'utenza e delle credenziali e delle relative scadenze. Si incarica inoltre di registrare gli accessi nei log di sistema, per questo motivo deve essere utilizzato per tutti e quattro i servizi `auth`, `password`, `account` e `session`.

L'uso di questo modulo implementa in sostanza il funzionamento classico di un sistema Unix così come era prima dell'uso di PAM, anche se sono presenti alcune piccole differenze e migliorie. Una differenza è che agli utenti senza password (quelli a cui si tolta la "`x`" dal secondo campo di `/etc/passwd`) viene comunque negato l'accesso, a meno di non aver passato l'opzione `nullok` come argomento. Un comportamento più vicino a quello classico, usato da molte distribuzioni, può essere ottenuto con `nullok_secure`, in cui si continua a garantire l'accesso senza password ma solo se questo avviene su uno dei terminali consentiti elencati in `/etc/securetty` (vedi sez. 4.3.5). Queste due opzioni hanno effetto solo se usate sul servizio `auth`.

Alcune estensioni vengono attivate con l'uso di altri argomenti, ad esempio si può richiedere la cifratura della password con algoritmi diversi usando le opzioni `md5`, `sha256`, `sha512`,[33] imporre una dimensione minima della stessa con l'argomento `min=n` dove `n` indica il numero di caratteri, o impedire il riuso delle ultime `n` password utilizzate con `remember=n`. Le vecchie password saranno mantenute in `/etc/security/opasswd`, così da evitarne il riutilizzo. Queste opzioni, attenendo al cambiamento della password, hanno effetto solo se indicate sul servizio `password`.

Oltre a `pam_unix` per reimplementare il comportamento classico di un sistema Unix sono disponibili una serie di altri moduli elementari che consentono di utilizzare attraverso PAM i vari file di configurazione della procedura di login illustrati in sez. 4.3.5. Ad esempio usare `pam_motd` sul servizio `session` consente di stampare il contenuto di `/etc/motd` all'inizio di una sessione, mentre sul servizio `auth` si possono usare `pam_nologin` per disabilitare l'accesso agli utenti normali se è presente `/etc/nologin` e `pam_securetty` per impedire l'accesso all'amministratore dai terminali non elencati in `/etc/securetty`.

Un altro modulo che consente di ottenere in maniera generica con PAM una delle funzionalità fornite una volta direttamente da `login` è `pam_limits`, che consente di imporre dei limiti su alcune delle risorse del sistema che un utente può utilizzare durante una sessione, come un numero massimo di processi contemporanei, numero di file aperti, quantità di memoria utilizzata, il tempo massimo di CPU, ed altre risorse più complesse. Le versioni di `login` precedenti all'uso di PAM, effettuavano direttamente l'impostazione dei limiti con il file `/etc/limits`, oggi non più utilizzato.

Il kernel infatti supporta la possibilità per qualunque processo di impostare una serie di limiti su alcune delle risorse che da esso verranno utilizzate. Questi limiti possono essere impostati solo all'interno del singolo processo,[34] ma vengono ereditati nella creazione di eventuali processi figli ed in questo modo una volta impostati su un processo iniziale (ad esempio la shell di login) possono essere mantenuti per tutti i comandi da essa eseguiti.

La shell consente di controllare i limiti sulle risorse con il comando interno `ulimit`, che oltre a mostrare la situazione corrente (con l'opzione `-a` che stampa i valori correnti dei limiti) permette anche di impostarne i valori. Questi, come avviene per le quote disco che tratteremo in sez. 6.4, prevedono la possibilità di avere due valori, un limite "*soffice*" (o *soft limit*, selezionabile con

[33] la configurazione di default prevede l'uso di `md5` per l'algoritmo MD5, gli altri sono algoritmi alternativi, e più sicuri, che possono essere utilizzati qualora siano supportati.

[34] per i dettagli sul funzionamento si può consultare sez. 8.3.2 di [GaPiL].

l'opzione `-S`) o un limite "*duro*" (*hard limit*, selezionabile con l'opzione `-H`). Per le altre opzioni, che permettono di indicare il limite ed il relativo valore, si consulti la pagina di manuale della shell.

È cura del kernel assicurarsi che un processo non possa utilizzare una risorsa per valori superiori a quelli indicati nel rispettivo limite "*soffice*", generando un errore quando questi vengono raggiunti. Il valore del limite "*duro*" serve invece come tetto massimo che può essere assegnato ad un limite "*soffice*". Un utente ordinario può imporsi dei limiti assegnando un valore sia al limite "*soffice*" che a quello "*duro*", il primo però non potrà mai superare quello del secondo. Inoltre una volta che un utente abbia impostato un limite "*duro*" il valore di quest'ultimo potrà solo essere ulteriormente ridotto, sempre che non si sia l'amministratore che può alzare i valori anche di quest'ultimo.

L'uso dei limiti consente di contrastare efficacemente i comportamenti inappropriati di quegli utenti che cercano di abusare delle risorse del sistema dando origine ad eventuali disservizi, o ad evitare tentativi espliciti di esaurimento delle risorse come le *fork bomb*, in cui si lanciano un numero indefinito di processi fino a bloccare completamente il sistema. Il problema è che i limiti devono essere imposti dall'amministratore solo a monte della procedura di collegamento al sistema; nel caso di un login da console questo potrebbe essere fatto usando il comando `ulimit` nel file `/etc/profile`, ma varrebbe solo per le shell che usano questo file, ed inoltre il limite verrebbe applicato in maniera indifferenziata.

La soluzione a questo problema viene fornita appunto dal modulo `pam_limits`, da inserire come `requisite` nelle regole associate al servizio `session` per i vari programmi che danno accesso al sistema. Il modulo si appoggia al file `/etc/security/limits.conf` all'interno del quale si possono impostare tutti i limiti che verranno attivati fin dall'inizio della sessione di lavoro, e mantenuti pertanto nel corso della stessa. Il file prevede una sintassi molto semplice dove al solito righe vuote o inizianti con "`#`" vengono ignorate; ciascuna riga è divisa in quattro campi separati da spazi, nella forma:

```
domain        type  item  value
```

Il primo campo, `domain`, indica a chi si applica la regola, vi si può specificare un nome utente, o un nome di gruppo (apponendovi il carattere "`@`"), o l'indicazione di un valore di default usando il carattere "`*`". Il secondo campo, `type`, specifica il tipo di limite e può essere `soft` o `hard` per indicare i rispettivi tipi, o "`-`" per indicare un limite sia *soft* che *hard*. Il terzo campo, `item`, indica a quale risorsa si vuole applicare il limite, il cui valore deve essere indicato nell'ultimo campo. Ciascuna risorsa viene identificata da una parola chiave, ed uno specifico intervallo ed unità di misura per il valore del limite; si sono riportate le più rilevanti in tab. 4.34, per l'elenco completo si consulti la pagina di manuale, accessibile con `man limits.conf`.

Si noti poi come oltre alle limitazioni sull'uso delle risorse da parte dei singoli processi, `pam_limits` consenta anche di impostare limiti relativi alla situazione generale del sistema, come quelli sul numero massimo di collegamenti contemporanei. Un esempio del contenuto di questo file potrebbe essere il seguente:

/etc/security/limits.conf

```
*               hard    nproc           1024
*               hard    nofile          1024
@student        hard    nproc           100
@student        hard    nofile          100
@student        -       maxlogin        10
```

Nome	Risorsa
fsize	massima dimensione utilizzabile per un singolo file.
nofile	massimo numero di file che un processo può aprire in contemporanea.
cpu	massimo tempo di CPU utilizzabile dal singolo processo.
nproc	numero massimo di processi che possono essere creati dallo stesso utente.
maxlogins	numero massimo di collegamenti contemporanei al sistema da parte dell'utente (valido solo per utenti normali).
maxsyslogins	numero massimo di collegamenti contemporanei al sistema (come somma di quelli tutti gli utenti) oltre il quale l'utente non avrà accesso.
nice	massimo valore di *nice* che può essere assegnato ad un processo.

Tabella 4.34: Nomi delle principali risorse che possono essere specificate come valore per il terzo campo di `/etc/security/limits.conf`.

Un primo esempio di come le funzionalità dei metodi tradizionali possono essere estese attraverso l'uso di PAM è quello del modulo `pam_cracklib`, da utilizzare nel servizio `password`. Il modulo consente di impostare delle politiche di gestione sul tipo di password usate dagli utenti, supportando una analisi di quelle da essi fornite nei confronti di un dizionario per evitare l'uso di parole comuni facili da indovinare. Consente così all'amministratore di imporre criteri che garantiscano una sufficiente complessità delle stesse per rendere più difficoltoso anche un attacco a dizionario.

L'uso del modulo fa sì che alla prima immissione di una password questa venga controllata per verificarne una sufficiente complessità sia facendo ricorso ad un dizionario che ad eventuali regole aggiuntive. Se il controllo viene passato allora verrà effettuata la seconda richiesta per conferma, altrimenti la password sarà rifiutata e all'utente sarà richiesto di immetterne una diversa con un breve messaggio esplicativo. Il modulo usa le librerie *cracklib*, da cui deriva il nome, per effettuare il controllo della password nei confronti di uno o più dizionari di parole comuni ed applica poi una serie di ulteriori controlli, come il fatto che la nuova password non sia troppo simile alla precedente, o troppo corta o già usata in precedenza come per `pam_unix`.

Il modulo prevede una serie di argomenti che consentono di controllare le condizioni aggiuntive. Con `minlen`, assegnato ad un intero positivo, si indica la lunghezza minima di una password; il default è 9, ma al numero di caratteri effettivo viene aggiunto anche un bonus di un punto per ciascun tipo di carattere diverso che si è messo nella password (numero, maiuscola, minuscola o punteggiatura) per cui si può superare il limite anche con un numero inferiore di caratteri. Un altro argomento che consente di agire sulla complessità richiesta è `difok` (anch'esso da assegnare ad un intero positivo) che richiede il numero minimo di caratteri che devono essere diversi dalla password precedente. Un esempio di uso di questo modulo è il seguente:

```
password  required pam_cracklib.so difok=3 minlen=10
```

Un secondo esempio di estensione delle funzionalità di controllo che tramite PAM possono essere applicate a qualunque programma, è quello del modulo `pam_listfile`. Questo modulo infatti consente di attivare un meccanismo di restrizione dell'accesso basato sul contenuto di

un file in cui si elenca la lista dei valori consentiti o proibiti di una certa proprietà associata alla sessione (username, terminale, macchina da cui ci si collega, ecc.). In questo modo diventa possibile applicare restrizioni aggiuntive, indipendenti fra loro, al singolo programma che usa il modulo.

Valore	Significato
`tty`	terminale di accesso.
`user`	nome dell'utente.
`rhost`	macchina remota.
`ruser`	utente remoto.
`group`	gruppo dell'utente.
`shell`	shell di login.

Tabella 4.35: Valori usabili per l'argomento `item` del modulo `pam_listfile`.

La proprietà della sessione che si vuole controllare nei confronti della propria lista si imposta con l'argomento `item`, da assegnare ad uno dei valori di tab. 4.35. Gli altri argomenti necessari al funzionamento del modulo sono `sense` cui assegnare rispettivamente i valori `allow` o `deny` a seconda che si voglia usare la lista per consentire o negare l'accesso e `file`, cui assegnare il pathname che identifica il file in cui mette l'elenco dei valori della proprietà indicata con `item`. La documentazione più dettagliata sul modulo ed i suoi argomenti può essere ottenuta dalla relativa pagina di manuale, accessibile con `man pam_listfile`.

4.4 Amministrazione sistemistica di una base di dati

Benché la gestione delle basi di dati sia una attività usualmente distinta dalla gestione sistemistica,[35] la diffusione dell'uso dei database relazionali comporta anche per l'amministratore di sistema la necessità di avere un minimo di nozioni al riguardo. Pertanto in questa sezione tratteremo le problematiche di gestione di un database esclusivamente dal punto di vista di un amministratore di sistema, in particolare per quanto riguarda la gestione sistemistica di server come PostgreSQL o MySQL, ed introdurremo alcune nozioni elementari sul linguaggio SQL.

4.4.1 Alcuni concetti base dei database relazionali

Benché la teoria dei database relazionali vada al di là di quanto interessa trattare in questo testo, vale la pena illustrare alcune nozioni di base che consentano quantomeno di avere una idea del quadro generale in cui si va ad operare quando si ha a che fare con queste problematiche, ed introdurre un minimo di terminologia che consenta di orientarsi nella documentazione.

Si tenga comunque presente che il concetto di base di dati è molto più generale di quello di database relazionale, tanto che oggi si sta assistendo ad un fiorire di database, detti *NoSQL*, che non adottano più questo modello, che è sostanzialmente basato sulla strutturazione dei dati in forma di tabelle.

[35] in genere a questo è dedicata la figura del DBA (*Data Base Administrator*) che richiede competenze specifiche sulle migliori modalità di gestione dei dati e della loro ottimizzazione, tutti argomenti che non prenderemo assolutamente in esame.

Per molti anni il vantaggio di avere un modello con una chiara rappresentazione matematica ed implementazioni ben funzionanti ha portato a cercare di ridurre tutti i problemi a questo modello, anche se esistono da molto tempo modelli alternativi, come quello ad albero usato da LDAP e quello chiave-valore usato da BerkleyDB. Oggi ci si sta rendendo conto delle sue inadeguatezze e si stanno sviluppando modelli alternativi, che sono più vicini al tipo di dati che si devono gestire ma la trattazione di questi argomenti va comunque al di là dello scopo di questo testo.

Semplificando molto una teoria piuttosto complessa, che non tratteremo neanche superficialmente, si può considerare un database relazionale come un insieme di tabelle di dati. Ogni tabella è costituita da un elenco di dati organizzati in righe (che vengono chiamate anche *record* o *tuple*), dove ciascuna riga individua un singolo oggetto nella tabella. Ciascuna riga è composta da una serie di campi detti *campi* o *attributi* (in inglese *field* o *attributes*), che vanno a costituire le colonne della tabella. Avendo in genere detti campi un significato preciso, vengono identificati da un nome, che viene a costituire la struttura della tabella stessa, definita appunto come l'elenco dei campi che ne costituiscono le righe.

In sostanza creare una tabella significa definire l'elenco dei campi che compongono le righe in essa contenute, associando a ciascuno di essi un nome identificativo ed il tipo di valore che può assumere, esistono ovviamente diversi tipi di valori possibili, ed anche il tipo di valore va a costituire la struttura di una tabella. Una riga sarà costituita dalla serie dei valori avuti da ciascuno dei suoi campi, alcuni dei quali possono anche essere assenti o indefiniti, nel qual caso si parla di valore "*nullo*" (il cosiddetto *null value*), che viene identificato specificamente nel linguaggio SQL, che vedremo in sez. 4.4.4, da una opportuna parola chiave (`NULL`). Il contenuto di una tabella è semplicemente costituito dall'insieme dei *record* che essa contiene, che non è previsto debba avere alcun ordinamento specifico.

Uno dei concetti principali nei database relazionali infatti è quello dell'uso delle cosiddette *chiavi* per effettuare le ricerche. Il concetto è di natura logica, si tratta in sostanza di usare i valori che hanno alcuni campi (sia presi singolarmente che eventualmente come insieme) per ottenere i *record* che hanno quel valore. In pratica le *chiavi* non sono altro che i campi sui quali si effettuano le ricerche, sui quali viene eseguita una opportuna indicizzazione in modo da renderle il più veloci possibile.

Quando una chiave consente di identificare in maniera univoca una riga della tabella, si parla di *chiave primaria* (o *primary key*) della tabella. Spesso queste *primary key* vengono generate automaticamente per essere sicuri dell'univocità, dato che alcuni campi non è detto abbiano questa proprietà. Questa caratteristica ne comporta delle altre, ad esempio una *primary key* non può avere valore nullo, non può essere modificata, e non può essere riutilizzata in caso di cancellazione della riga corrispondente.

Il nome di database relazionale dato a questo tipo di strutturazione dei dati deriva dal fatto che una tabella può essere vista come una "*relazione*"[36] fra i dati che la costituiscono, tanto che talvolta questo (o l'inglese *relation*) è un nome alternativo con cui vengono indicate le tabelle. La potenza di un database relazionale sta nel fatto che grazie alle chiavi primarie si possono definire delle relazioni fra tabelle diverse (in realtà l'argomento sarebbe molto più ampio, ma la sua trattazione va al di là di quanto appropriato per questa introduzione.)

[36] il concetto deriva direttamente dall'algebra ed ha una fondazione matematica rigorosa, per approfondimenti si consulti ad esempio `http://it.wikipedia.org/wiki/Algebra_relazionale`.

Si può cioè avere una tabella di autori con i dati relativi agli stessi, ad esempio nome e cognome ed età, ed una di libri con rispettivi dati, ad esempio titolo ed editore ed anno di pubblicazione, e metterle in *relazione* fra loro usando ad esempio come campo autore nella tabella dei libri il valore di una chiave primaria della tabella degli autori, in modo che questi possano essere identificati univocamente anche se i loro dati sono mantenuti in maniera indipendente su una diversa tabella. In questo caso il valore del campo autore nella tabella dei libri è quello che si chiama una *chiave esterna* o *foreign key*.

In genere si indica con RDBMS (*Relational Database Management System*) il software che consente di operare su dati mantenuti in un database relazionale, anche se spesso si finisce semplicemente per chiamarlo *database*. Un RDBMS serve ad operare in maniera veloce ed efficiente sui dati attraverso il linguaggio SQL (che vedremo più avanti in sez. 4.4.4), esso deve pertanto gestire da una parte l'interpretazione del linguaggio e la esecuzione dei relativi comandi, dall'altro l'opportuno salvataggio dei dati e la loro integrità.

In genere un software RDBMS consente di gestire più basi di dati, vale a dire tanti database distinti, con dati diversi, ed oltre a consentire la manipolazione degli stessi con il linguaggio SQL fornisce in genere anche il supporto per poter avere gli opportuni controlli di l'accesso e la possibilità di creare utenti e assegnare loro diverse capacità relative all'uso dei dati stessi.

4.4.2 Configurazione e gestione di base di PostgreSQL

PostgreSQL è uno dei più completi, stabili ed efficienti RDBMS disponibili, che supporta numerose funzionalità avanzate come transazioni, subselect, trigger, viste, *foreign key* ed integrità referenziale che gli consentono di poter sostituire il ben più famoso (e costoso) Oracle, con il quale è in grado di lottare ad armi pari nella gran parte dei casi.

Non tratteremo l'installazione del programma, in quanto questo viene fornito già pacchettizzato dalle principali distribuzioni,[37] e trattandosi di un server di grande importanza per la propria infrastruttura è sempre opportuno utilizzare una versione pacchettizzata per la quale la propria distribuzione garantisca il supporto per gli aggiornamenti. Anche PostgreSQL, come per la gran parte degli RDBMS, adotta un modello client/server, in cui l'accesso ai dati viene fornito da un programma server (nel caso `postmaster`) cui si possono fare richieste tramite opportuni client, pertanto a seconda dei casi si dovranno installare i relativi pacchetti.

La configurazione (del server) viene mantenuta all'interno di una directory dipendente dall'installazione. Quando il programma viene installato dai sorgenti l'approccio degli sviluppatori è quello di mantenere tutti i file, dati e configurazioni, nella directory di installazione; questo però contrasta con le direttive del *Filesystem Hierarchy Standard*, per cui le versioni pacchettizzate possono fare scelte diverse. Ad esempio nel caso di Debian i file di configurazione vengono riportati in una directory sotto `/etc/` nella forma `/etc/postgresql/N.N/main` (dove `N.N` è il numero della versione, ad esempio 7.4 o 8.3).[38] Il file di configurazione principale è `postgresql.conf` che contiene le direttive di configurazione nella classica forma `chiave=valore`, con la solita convenzione di ignorare righe vuote e tutto ciò che segue il carattere "`#`".

[37] nel caso di Debian sono addirittura pacchettizzate le diverse versioni stabili, e per Lenny sono disponibili sia la versione 7.4 che la 8.3.

[38] la scelta della sottodirectory con la versione è dovuta al supporto contemporaneo di diverse versioni del programma.

Non staremo qui a trattare i dettagli della configurazione del server che, riguardando principalmente le modalità di gestione ed utilizzo dei dati e le relative ottimizzazioni, vanno al di là dello scopo introduttivo di questo paragrafo, se però si vuole che il server possa essere raggiunto via rete e non soltanto in locale occorrerà inserire la direttiva:[39]

```
listen_addresses = '*'
```

Il secondo file di interesse per la configurazione di base di PostgreSQL è `pg_hba.conf`, che definisce le regole per il controllo di accesso e l'autenticazione da parte dei client. Il file è composto da una serie di righe ciascuna delle quali indica una modalità di accesso ed autenticazione, e la prima che corrisponde alle modalità di accesso che si stanno usando verrà utilizzata per l'autenticazione.

Le righe sono composte di un serie di campi separati da spazi vuoti, il primo campo indica la modalità di connessione al server, ed in genere si usa o il valore `local` per usare un socket locale, o il valore `host` per una connessione via rete. I due campi successivi sono identici indipendentemente dal valore del primo campo: il secondo indica il nome del database a cui si applica il criterio di accesso, ed il valore più comune è `all` per indicarli tutti; il terzo indica l'utente, e di nuovo il valore `all` indica l'utente generico.

Nel caso si sia usato come criterio di accesso `host`, i campi successivi servono per determinare gli indirizzi di rete consentiti, si può specificare il singolo indirizzo e rete in notazione CIDR, o in due campi separati indirizzo e maschera di rete (al solito per queste notazioni si fa riferimento ai concetti spiegati in sez. 7.2). Si faccia attenzione, nel caso di singolo indirizzo IP, a specificarlo comunque come rete in una delle due forme.

Nel caso si sia usato come criterio di accesso `local`, non è necessario specificare nessun indirizzo e può passare direttamente agli ultimi due campi che indicano il metodo di autenticazione e le eventuali relative opzioni. In genere per le connessioni via rete si usa il metodo `md5` che indica che l'utente deve essere autenticato con il valore della password cifrata mantenuto dal server stesso. Nel caso di `local` si può far ricorso al metodo `password`, che richiede l'invio di una password in chiaro, o `ident sameuser` che richiede che il client sia posto in esecuzione da un utente unix con lo stesso nome usato dal rispettivo utente del database.[40]

Un estratto di questo file, preso dalla versione installata di default da una Debian Lenny, è il seguente, sarà necessaria una modifica soltanto nei casi in cui si vogliano consentire accessi da altre reti, aggiungendovi una opportuna riga `host`:

pg_hba.conf

```
# Database administrative login by UNIX sockets
local   all         postgres                          ident sameuser

# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD

# "local" is for Unix domain socket connections only
local   all         all                               password
# IPv4 local connections:
host    all         all         127.0.0.1/32          md5
# IPv6 local connections:
host    all         all         ::1/128               md5
```

[39] il default è ascoltare soltanto sull'indirizzo locale (per i concetti relativi alle reti si rimanda al solito a sez. 7.2).

[40] questa è la modalità usata da Debian per dare accesso all'utente `postgres` usato per la gestione generale del database, creando un utente omonimo nel sistema, a cui viene anche assegnata la proprietà dei file del server.

Il comando principale per operare su un database PostgreSQL è `psql`, che invoca il programma client con cui ci si può connettere al server per le eseguire operazioni a riga di comando. Se usato interattivamente le istruzioni da eseguire verranno lette da un prompt sul terminale, ma lo si può usare anche in modalità non interattiva facendo leggere le istruzioni da un file o comunque con una redirezione dello *standard input*.

Opzione	**Significato**
`-f` *file*	legge i comandi da eseguire file passato come parametro.
`-c` *cmd*	esegue il comando passato come parametro (da inserire come stringa) ed esce immediatamente.
`-o` *file*	scrive i risultati sul file passato come parametro.
`-l`	elenca i database disponibili sul server.
`-h` *host*	si collega all'indirizzo di rete passato come parametro.
`-p` *port*	si collega alla porta passata come parametro.
`-U` *user*	si collega con l'utente passato come parametro.
`-W`	forza la richiesta della password utente prima di collegarsi.

Tabella 4.36: Principali opzioni del comando `psql`.

Il comando richiede come argomenti il nome del database su cui si vuole operare e di seguito il nome dell'utente che si vuole utilizzare. Se non si specifica un database si potranno usare solo i comandi di servizio che non fanno riferimento ai dati, e se non si specifica un nome utente verrà usato l'*username* di chi esegue il comando. Il default è usare una connessione locale, ma ci si può collegare via rete indicando esplicitamente usando l'opzione `-h` per indicare la macchina a cui collegarsi (per indirizzo IP o nome a dominio). Le opzioni principali sono riassunte in tab. 4.36, per la descrizione completa al solito si faccia riferimento alla pagina di manuale.

Una volta effettuata la connessione al server, ed a meno che non si sia indicato un file da cui leggere i comandi con l'opzione `-f` o con la redirezione dello *standard input*, il comando mette a disposizione un prompt su cui scrivere interattivamente le proprie istruzioni dopo aver stampato un messaggio di benvenuto con una legenda dei *comandi di controllo* più importanti. Si tratta di comandi che attengono alla gestione del server stesso, e non alle operazioni sui dati per i quali si deve usare il linguaggio illustrato in sez. 4.4.4 e si distinguono dalle istruzioni previste dal linguaggio SQL in quanto sono preceduti dal carattere "`\`".[41] Un elenco degli stessi può essere ottenuto con "`\?`", mentre con "`\h`" si ottengono invece i comandi SQL. Per uscire dalla riga di comando si può usare "`\q`". Gli altri comandi di controllo più importanti sono riportati in tab. 4.37.

Come accennato PostgreSQL fornisce anche un sistema di controllo degli accessi basato sulla presenza di utenti a cui possono essere assegnati diversi permessi sia sulle possibilità di intervenire sui singoli database gestiti dal server, sia in termini di amministrazione generale del server stesso. Gli utenti vengono mantenuti su delle tabelle interne al database, ma per semplificarne la gestione sono stati creati due comandi appositi. In realtà entrambi i comandi non sono che *wrapper* per i comandi SQL `CREATE ROLE` e `DROP ROLE` (nel gergo di PostgreSQL gli utenti sono chiamati *ruoli*), e possono essere usati anche in questo modo dall'interno del client.

[41] in realtà alcuni di essi sono solo abbreviazioni predefinite dei comandi SQL necessari ad ottenere le informazioni volute.

Comando	Significato
\c	si connette a un database (da passare come parametro).
\q	esce dal programma.
\l	stampa l'elenco dei database presenti.
\!	esegue il comando di shell passato come parametro.
\d	stampa l'elenco delle tabelle del database corrente.
\u	stampa l'elenco degli utenti.

Tabella 4.37: Principali comandi di controllo di `psql`.

Il primo è `createuser` e si utilizza per creare un nuovo utente, il comando richiede semplicemente come argomento un nome utente, che verrà creato. In realtà, dato che il comando non è altro che un *wrapper* per l'esecuzione di comandi SQL, la creazione può avvenire soltanto se l'utente di sistema che si usa per eseguire questo comando è in grado di collegarsi al database e l'utente PostgreSQL a cui corrisponde ha sufficienti privilegi (che sono quelli di amministratore o la capacità di creare altri *ruoli*).

Il comando prevede che si possa specificare se il nuovo utente sarà un amministratore (*superuser*) con l'opzione `-s`, se può creare nuovi database con l'opzione `-d`, se può creare nuovi utenti con l'opzione `-r`, ma se non si specifica nulla queste indicazioni verranno chieste interattivamente. Si può inoltre richiedere l'impostazione di una password per il nuovo utente, che verrà richiesta interattivamente, con l'opzione `-P`. Infine il comando consente anche di eseguire le operazioni collegandosi ad un database remoto con `-h`, e supporta le stesse opzioni di `psql` per indicare i parametri di connessione (utente, password, ecc.) riportate nella seconda parte di tab. 4.36.

Il secondo comando di gestione degli utenti è `dropuser` che consente di cancellare un utente, da specificare come argomento, anche in questo caso si può effettuare l'operazione su un server remoto con le stesse opzioni riportate nella seconda parte di tab. 4.36. Anche in questo caso si tratta di un *wrapper* alle relative istruzioni SQL, e per poterlo usare senza dover eseguire una connessione esplicita occorrerà eseguirlo con un utente di sistema in grado di collegarsi direttamente al database.

Una seconda coppia di comandi di gestione, `createdb` e `dropdb` consentono rispettivamente di creare un nuovo database o rimuoverne uno esistente (con la relativa perdita del contenuto). Entrambi prendono le opzioni della seconda parte di tab. 4.36 per collegarsi ad un database remoto. Inoltre `createdb` supporta l'uso dell'opzione `-O` che consente di assegnare la proprietà del nuovo database ad un utente diverso da quello con cui si esegue il comando, ovviamente posto che si abbiano i privilegi necessari per farlo.

Infine uno dei comandi più importanti per la gestione sistemistica di PostgreSQL è `pg_dump`, che consente di esportare di tutti i dati di un database in un file di testo contenente tutte le istruzioni SQL necessarie a ricreare su un database vuoto lo stesso contenuto di quello esportato usando `psql`, quello che appunto viene chiamato un "*dump*" del database, fornendo così il supporto sia per un backup completo dei dati che per la conversione degli stessi in un formato portabile che può essere riutilizzato anche con diverse versioni del software.

Se infatti è sempre possibile effettuare un backup dei dati andando a salvare i file binari su cui PostgreSQL salva le informazioni, questi non è detto siano in uno stato consistente, e soprattutto non è garantito, a meno di non usare esattamente la stessa versione sulla stessa piattaforma hardware, che essi possano essere riutilizzati direttamente. Un *dump* con `pg_dump`

assicura invece la riutilizzabilità dei dati anche fra diverse versioni del server ed è sempre la modalità preferita per effettuare i salvataggi dei dati di un database, considerazione che resta vera qualunque sia il software che si usa (vale ad esempio anche per MySQL).

Il comando richiede come argomento il nome del database di cui si vuole effettuare il "*dump*", e supporta una serie di opzioni di controllo per le modalità di creazione dello stesso. Ad esempio con `-c` si richiede che vengano inseriti nel *dump* i comandi per cancellare il database prima di ricrearlo, con `-f` si indica un file di uscita invece dello *standard output*. Sono inoltre supportate le opzioni per gestire la connessione verso un server già riportate nella seconda parte di tab. 4.36 ed una serie di ulteriori opzioni per eseguire il salvataggio solo di alcune parti di dati.

Analogo a `pg_dump` e con buona parte delle stesse opzioni, fra cui le due citate in precedenza e quelle per il collegamento al server, è `pg_dumpall` che invece di eseguire il *dump* di un database specifico lo effettua per tutti i database presenti su un server, semplificando la gestione amministrativa dei backup. Una opzione specifica è `-l` che consente di indicare il nome database di amministrazione interno su cui vengono mantenuti i dati globali del server, come l'elenco degli altri database e degli utenti. In genere non è necessario indicarlo in quanto vengono il default utilizza i nomi (`postgres` o `template1`) che vengono creati automaticamente nell'installazione del servizio.

Occorre infine menzionare la possibilità, necessaria quando si vogliono usare questi due ultimi comandi all'interno di uno script di backup eseguito in batch che richieda l'uso di una connessione esplicita con autenticazione, di usare il file `.pgpass` per evitare di dover immettere le password sul terminale. Infatti benché sia possibile usare delle variabili di ambiente per le credenziali di connessione, queste non sono sicure perché possono essere rilette da chiunque con `ps`. Il file ha un formato generico che prevede 5 campi separati da "`:`" nella forma:

```
hostname:port:database:username:password
```

il cui significato è autoesplicativo, e si può usare il carattere jolly "*" quando non si intende specificare un valore per uno dei primi quattro campi. Il file deve essere mantenuto nella home directory dell'utente che esegue i comandi e per garantire la sicurezza delle informazioni viene ignorato se ha permessi di accesso per il gruppo o gli altri.

4.4.3 Configurazione e gestione di base di MySQL

MySQL è il *database server* libero (disponibile con licenza GPL) più diffuso,[42] usato da innumerevoli applicazioni Web, tanto da costituire la terza lettera dell'acronimo *LAMP*, che sta ad indicare, a seconda delle preferenze di linguaggio del programmatore, la lista Linux, Apache, MySQL, Perl/PHP/Python. Le sue caratteristiche sono la leggerezza e la semplicità di amministrazione, ma ha solo una parte delle funzionalità disponibili con PostgreSQL, anche se nella maggior parte dei casi queste sono sufficienti.

Come per PostgreSQL non tratteremo le modalità di installazione, essendo già disponibili i relativi pacchetti in tutte le principali distribuzioni.[43] Anche MySQL funziona adottando un

[42] anche grazie fatto di aver potuto contare sulla promozione da parte dell'azienda che ne deteneva il codice e che vendeva licenze proprietarie per chi non voleva sottostare alle condizioni della GPL; oggi il codice è divenuto proprietà di Oracle, cosa che ha fatto sorgere qualche dubbio relativo al suo futuro sviluppo, essendo un chiaro concorrente dell'omonimo prodotto proprietario.

[43] nel caso di Debian il pacchetto si chiama `mysql-server` e fa parte della distribuzione ufficiale.

modello client/server, in cui l'accesso ai dati viene eseguito tramite un programma server (nel caso `mysqld`) cui si possono fare richieste con i relativi client, per l'uso dei quali si dovranno installare i relativi pacchetti.

Il file di configurazione del server è `/etc/mysql/my.cnf`, che contiene le direttive che ne controllano il funzionamento e che consentono di ottimizzarne le prestazioni. Il formato è quello dei file INI di Windows, diviso in sezioni il cui nome è racchiuso fra parentesi quadre e con direttive di configurazione nella classica forma `chiave=valore`, anche in questo caso vale la solita convenzione di ignorare righe vuote e tutto ciò che segue il carattere `#`.

Per lo scopo introduttivo di questo paragrafo le sole direttive di uso sistemistico che possono interessare sono quelle relative alle modalità con cui il server può essere interrogato via rete, che si trovano nella sezione marcata `[mysqld]`. Come per PostgreSQL il default è ascoltare soltanto su `localhost` (per i concetti relativi alle reti si rimanda al solito a sez. 7.2) e se lo si vuol rendere raggiungibile non solo in locale occorrerà inserire la direttiva:

```
bind_address = 0.0.0.0
```

Le altre direttive consentono di specificare le directory ed i file usati dal programma e l'utente per conto del quale deve essere eseguito, che nelle versioni pacchettizzate già presenti nelle principali distribuzioni sono già preimpostate a valori opportuni; l'unica altra che può essere significativa è `port` che specifica la porta su cui il programma si può porre in ascolto. Ulteriori importanti direttive di questa sezione sono quelle attinenti alle dimensioni di memoria da assegnare a buffer e cache, la cui trattazione però va oltre lo scopo di questo testo.

Il comando principale per usare il database è `mysql` che invoca il programma con cui ci si può connettere al database. Se usato interattivamente le istruzioni devono essere inviate dal prompt sul terminale, ma il comando può essere invocato anche in modalità non interattiva, facendogli leggere le istruzioni da un file o con una redirezione dello *standard input*. Il comando richiede in genere che si specifichi un utente con l'opzione `-u` e se per questo utente è prevista una password deve essere usata l'opzione `-p` per richiederne l'immissione. Le altre opzioni principali sono riportate in tab. 4.38, per l'elenco completo si consulti la pagina di manuale.

Opzione	**Significato**
`-e` *cmd*	esegue il comando passato come parametro (da inserire come stringa) ed esce immediatamente.
`-h` *host*	si collega all'indirizzo di rete passato come parametro.
`-P` *port*	si collega alla porta passata come parametro.
`-u` *user*	si collega con l'utente passato come parametro.
`-p`	forza la richiesta della password utente prima di collegarsi.

Tabella 4.38: Le principali opzioni del comando `mysql`.

Il comando consente di specificare un argomento che indica il database a cui ci si vuole collegare (fra quelli presenti sul server), perché questo sia possibile occorre però che l'utente con il quale ci si collega abbia gli opportuni privilegi di accesso. L'installazione di default prevede che sia sempre presente un utente amministrativo generale (in genere viene usato il nome `root` come per Unix) che ha accesso a tutti i database presenti.

Se non si specifica nessun argomento si accederà alla riga di comando senza essere collegati a nessun database e si potrà eseguire il collegamento in una fase successiva con il comando

interno `use`; ad esempio per collegarsi allo speciale database interno `mysql` che contiene tutti i dati relativi alla gestione amministrativa del server si dovrà eseguire il comando:

```
mysql> use mysql
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

Come per `psql` anche la riga di comando di `mysql` consente di inviare specifici comandi di controllo preceduti dal carattere "\", i principali dei quali sono riportati in tab. 4.39, ma in questo caso ad essi sono in genere associate anche delle esplicite parole chiave. Inoltre

Comando		**Significato**
`\u`	`use`	si connette a un database (da passare come parametro).
`\q`	`quit`	esce dal programma.
`\!`	`system`	esegue il comando di shell passato come parametro.
`\h`	`help`	stampa un riassunto dei comandi o la sintassi del comando passato come argomento.

Tabella 4.39: Principali comandi di controllo di `mysql`.

A differenza di PostgreSQL non sono previsti comandi specifici per creare utenti, per i quali si deve usare esplicitamente il linguaggio SQL per operare sulla tabella `user` del database riservato `mysql` che contiene tutti i dati relativi alla gestione amministrativa del server.[44]

I comandi SQL in questione sono due, il più semplice è `CREATE USER` che si limita alla creazione dell'utente, senza nessuna assegnazione di permessi. Il comando richiede che si specifichi come ulteriore argomento un nome utente. Questi nel caso di MySQL hanno un formato specifico nella forma:

```
'username'@'hostname'
```

(gli apici sono necessari) in quanto ogni utente è associato anche ad una macchina da cui questo può accedere via rete al server (si tratta dei campi `user` e `host` della citata tabella `user`), per cui se si vuole dare un accesso generico senza restrizioni relative alla macchina di partenza, occorre specificarlo come `'username'@'%'` usando il carattere jolly "`%`" del linguaggio SQL.

Se oltre a creare l'utente si intende pure assegnargli dei diritti di accesso occorre invece utilizzare il comando SQL `GRANT`, seguito dall'indicazione della lista dei privilegi come elenco di nomi separato da virgole (ad esempio `SELECT,INSERT,UPDATE,DELETE,CREATE,INDEX,ALTER, ...`) oppure con l'indicazione generica `ALL PRIVILEGES`.

A questo deve seguire come argomenti della parola chiave `ON` l'indicazione del database o delle singole tabelle cui i privilegi si applicano specificati come elenco separate da virgole nella forma `nomedb.nometabella`, con la possibilità di usare la forma `nomedb.*` per indicare un intero database. Per indicare l'utente, che verrà creato se non esistente, si deve poi aggiungere la parola chiave `TO` seguita dal nome col formato appena illustrato.

Infine per l'impostazione di una password di accesso, che verrà mantenuta in forma cifrata nel campo `password` della solita tabella `user`, entrambi i comandi prevedono l'uso di una ulteriore

[44] in realtà anche PostgreSQL mantiene i dati su una tabella interna, ma fornisce alcuni comandi che eseguono direttamente su di essa le operazioni necessarie alla gestione degli utenti senza dover ricorrere all'SQL.

istruzione SQL `IDENTIFIED BY` da aggiungere in coda al comando, con la password specificata in chiaro fra virgolette. In questo caso un esempio val più di mille parole, per cui si potranno creare degli utenti con una sintassi del tipo:

```
mysql> CREATE USER 'user1'@'localhost' IDENTIFIED BY 'pass1';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT ALL PRIVILEGES ON db.* TO 'user2'@'%' IDENTIFIED BY 'pass2';
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Nel caso di creazione di un database invece non è necessario ricorre all'uso di comandi dedicati SQL (anche se questi comunque sono presenti) e si può ricorrere al comando di amministrazione generale di MySQL `mysqladmin`, che viene usato per anche per altre operazioni amministrative sul server. Il comando richiede che si specifichi l'operazione da eseguire come primo argomento, seguita da eventuali argomenti successivi e richiede la connessione al server con le stesse opzioni di `mysql`, illustrate nella seconda parte di tab. 4.38.

Per creare un database si deve usare l'operazione `create`, seguita dal nome che si vuole dare al database stesso. Questo potrà essere anche cancellato completamente con l'operazione `drop`, sempre seguita dal nome. Oltre a queste due, che sono le operazioni più comuni, `mysqladmin` può essere usato per fermare e riavviare i processi del database o per reimpostare la password dell'utente che si è usato per collegarsi al server, per i dettagli si consulti la pagina di manuale.

Infine anche per MySQL è presente un opportuno comando di esportazione per i contenuti di un database in istruzioni SQL registrate su un file di testo, `mysqldump`. In questo il comando prende come argomento il nome di un database seguito eventualmente da una lista di tabelle se si vuole eseguire un *dump* solo di alcune di esse. Il comando riconosce inoltre l'opzione `--all-database` che esegue il *dump* di tutti i database presenti sul server.

Si tenga presente che per l'uso di `mysqldump` negli script di backup a differenza di PostgreSQL, per il quale si può avere un accesso senza password usando un utente dedicato, è comunque necessario autenticarsi e fornire una password, ed anche in questo caso, per gli stessi motivi di sicurezza già detti, la modalità suggerita è quella di utilizzare il file di configurazione delle opzioni di MySQL nella home dell'utente che esegue il backup che in questo caso è `.my.cnf`.

Il formato del file riprende sempre quello dei file INI ed ha uno scopo molto più generale in quanto serve da file di personalizzazione delle configurazioni generico e può contenere direttive che riguardano anche il server, ma per lo scopo specifico è sufficiente usare inserire la password che si vuole venga utilizzata in una direttiva `password` nella sezione `client`. In sostanza si dovrà avere un contenuto del tipo:

.my.cnf
```
[client]
password=pwdsupersecret
```

4.4.4 Nozioni elementari di SQL

Il linguaggio SQL (*Structured Query Language*) nasce, come dice il nome, come linguaggio per eseguire interrogazioni su una base di dati relazionale. Benché la sua conoscenza attenga più alle

competenze di un programmatore o di un amministratore di database, è opportuno anche per un amministratore di sistema padroneggiare almeno le istruzioni più semplici di questo linguaggio, data la sempre maggior diffusione dell'uso di database SQL da parte di molti programmi.

Sta infatti diventando sempre più comune, anche grazie alla diffusione di programmi come SQLite che consentono di mantenere un database relazionale su semplici file, memorizzare dati di gestione (e talvolta anche configurazioni) di un programma su un database di questo tipo per sfruttarne la velocità di accesso nelle ricerche. In questo caso molto spesso un amministratore di sistema si trova nelle condizioni di dover intervenire ed eseguire delle modifiche direttamente su tali dati, e per questo è necessaria una conoscenza delle basi del linguaggio SQL.

Purtroppo, nonostante esista un certo livello di standardizzazione,[45] il linguaggio SQL presenta innumerevoli varianti ed estensioni, nel nostro caso ci limiteremo ad esaminare le forme più elementari che sono sostanzialmente le stesse nelle varie implementazioni. Inoltre quanto segue non è da considerarsi minimamente una introduzione sistematica o formale a questo linguaggio, ma soltanto un riassunto delle principali istruzioni ad uso di un amministratore di sistema, orientate alla realizzazione di specifiche operazioni come cercare informazioni o eseguire modifiche elementari trascurando problematiche complesse come transazioni, atomicità, integrità referenziale ecc.

Il linguaggio prevede l'uso di una serie di istruzioni che verranno eseguite sul server per operare sui dati, dotate o meno di argomenti specifici, con l'uso come separatore dello spazio e quello della virgola qualora gli argomenti siano delle liste. In genere i valori numerici ed i nomi delle colonne si usano direttamente mentre eventuali valori testuali devono essere sempre immessi fra apici singoli.

Ogni comando SQL deve poi essere sempre terminato dal carattere "`;`" senza il quale l'uso del tasto di invio si limita a portare sulla riga successiva dove si può continuare a scrivere come se si fosse aggiunto uno spazio. Inoltre benché nella sintassi di SQL le istruzioni siano case insensitive, per convenzione si tende a scriverle sempre in lettere maiuscole, per distinguerle dai relativi argomenti e loro valori.

Il primo passo per l'uso del linguaggio è quello di "*collegarsi*" al proprio database per eseguire i relativi comandi dalla console fornita dallo stesso. Questo avviene, per i veri e propri *database server*, con una connessione (anche di rete) eseguita tramite un apposito programma client o, come nel caso di SQLite, con un programma che opera direttamente sui file dei dati. La sintassi per effettuare il collegamento varia ovviamente da programma a programma, ed abbiamo visto nelle sezioni precedenti come fare per PostgreSQL e MySQL. Nel caso si usi SQLite dato che ogni file è un database, basterà eseguire un comando come `sqlite3 dbfile.sqlite`.

Una delle operazioni più comuni da effettuare su un database è quella relativa alla ricerca dei dati. Trattandosi di database relazionale la struttura dei dati è sempre in forma di tabelle, ed i dati devono essere cercati all'interno delle stesse. Purtroppo non esiste una istruzione generica SQL per ottenere un elenco delle tabelle di un database, ed ogni programma usa in genere comandi diversi.

In realtà questi si riducono spesso alla ricerca dei dati all'interno di una tabella di gestione interna, ma spesso questa ricerca viene mascherata/semplificata fornendo alcuni comandi specifici della sintassi della propria versione di SQL o del client usato per il collegamento. Esempi

[45] il primo standard, realizzato dalla ANSI, è denominato SQL-86, ad esso sono seguite diverse revisioni, per un riassunto si legga `http://en.wikipedia.org/wiki/SQL#Standardization`.

di questi comandi sono "`\dt`" in PostgreSQL, "`show tables;`" in MySQL e "`.tables`" in SQLite, che consentono di ottenere un elenco delle tabelle presenti nel database su cui si sta operando.

L'istruzione generica di ricerca dei dati è `SELECT`, che nella forma più semplice deve essere seguita dalla lista delle colonne di cui si vogliono ottenere i valori in forma di elenco di nomi separate da virgole, o alternativamente, per ottenere i dati di tutte le colonne, dal carattere "`*`", a questa indicazione deve seguire l'istruzione `FROM`, seguita dal nome della tabella su cui si vuole effettuare la ricerca, con una sintassi generica che viene usata anche da altre istruzioni del linguaggio. Pertanto una delle modalità più comuni per esaminare il contenuto di una tabella è eseguire il comando:

```
SELECT * FROM tablename;
```

qualora invece si vogliano avere soltanto i valori di alcune colonne si dovrà inserire l'elenco delle stesse, con un comando del tipo:

```
SELECT column1,column2 FROM tablename;
```

L'uso delle due istruzioni precedenti comporta la stampa del contenuto di tutte le righe (completo o solo per i valori delle colonne richieste) contenute nella tabella. Se questa è grande si rischia di essere annegati nei risultati, pertanto diventa utile poter richiedere un intervallo di dati. Per questo la sintassi di SQL per `SELECT` prevede la possibilità di aggiungere la istruzione `LIMIT` che consente di indicare un numero di righe da restituire, sintassi che si può usare anche per altre istruzioni che prevedono di operare su un numero qualunque di righe.

Se non si specifica altro verranno restituite le prime righe della tabella, ma si può indicare un diverso punto di partenza usando la ulteriore istruzione `OFFSET` che indica una riga di partenza, inoltre molti programmi supportano anche una notazione compatta che consente di indicare direttamente con `LIMIT` un punto di partenza ed un numero di righe indicando l'argomento con due numeri nella forma `offs,rows`. Se allora si vogliono vedere le prime 10 righe di una tabella si dovrà usare un comando del tipo:

```
SELECT * FROM tablename LIMIT 10;
```

Una modalità diversa per restringere l'elenco dei risultati è quella di ricorrere all'istruzione `WHERE` che consente di selezionare le righe restituite in base ad una condizione sui valori assunti da una qualunque colonna, che in genere viene espressa nella forma più elementare come `nomecolonna='valore'` per richiedere la corrispondenza ad un certo valore. Pertanto per ottenere le righe in cui `column2` ha il valore `value` si dovrà usare un comando del tipo:

```
SELECT * FROM tablename WHERE column2='value';
```

La forma illustrata nel precedente esempio è quella più semplice in cui si richiede una semplice uguaglianza, ma sono supportate anche sintassi più elaborate, con operatori come "`>`", "`>=`", "`<`", "`<=`" e la parola chiave `LIKE` che consente di indicare un pattern analogo a quello del *filename globbing* dove il ruolo del carattere jolly viene però volto da "`%`", invece che dall'asterisco.

Sono poi supportate anche la negazione di una condizione con la parola chiave `NOT` e combinazioni logiche di più condizioni con le parole chiave `OR` e `AND` e l'uso delle ordinarie parentesi tonde. Si tenga presente che questa sintassi dell'istruzione `WHERE` è generica, e si ritrova identica

anche nella sintassi di altre istruzioni, come DELETE e UPDATE, dove come per SELECT viene usata per selezionare le righe su cui operare con un opportuno criterio di corrispondenza.

L'ultima estensione della sintassi di SELECT che vedremo è quella che consente di ottenere una lista di risultati opportunamente indicizzata. Per questo si deve ricorrere alla ulteriore istruzione ORDER BY che consente di richiedere un ordinamento dei dati sulla base dei valori dei campi indicati di seguito, e se ne possono specificare più di uno per avere ordinamenti progressivi su più campi. Inoltre si può chiedere un ordinamento discendente o ascendente (ma quest'ultimo è il default), rispettivamente con le ulteriori parole chiave DESC e ASC. Un esempio in questo caso sarebbe qualcosa del tipo:

```
SELECT * FROM tablename ORDER BY column2;
```

La seconda operazione amministrativa che capita di dover effettuare molto spesso su un database è quella di modificare il valore di un dato in una qualche tabella. In questo caso l'istruzione da usare è UPDATE seguita dal nome della tabella che su cui si intende operare, a cui poi deve seguire l'istruzione SET per indicare su quale colonna si vuole operare e quale è il valore da impostare, che dovrà essere indicato nella forma nomecolonna='valore'. In genere il linguaggio consente di modificare in contemporanea i valori di più colonne, semplicemente indicando più assegnazioni come la precedente separate da virgole.

L'uso di queste due istruzioni è sufficiente per avere un comando SQL corretto, ma l'effetto normalmente non è quello desiderato, dato che in questo caso si cambierebbero i valori per tutte le righe della colonna, cosa che raramente è quello che si vuole ed è invece un errore abbastanza frequente. Per questo anche se il resto è opzionale nella sintassi, il comando in genere deve essere completato con la ulteriore istruzione WHERE che indichi la riga o le righe su cui si vuole operare. Per questo il comando tipo che consente di cambiare il valore di un dato mantenuto nella colonna column1 per la riga in cui si ha uno specifico valore di column2 è qualcosa del tipo:

```
UPDATE tablename SET column1='newvalue' WHERE column2='identifier';
```

Ma si rammenti che anche in questo caso se si vuole cambiare il valore di una unica riga si deve essere sicuri che la selezione effettuata con WHERE dia un risultato unico, perché altrimenti verranno cambiati i valori di column1 per tutte le righe che corrispondono alla condizione su column2, si può però riutilizzare l'istruzione LIMIT per porre un limite sul numero di righe che verranno modificate.

Con UPDATE si possono modificare i valori presenti in una tabella, ma non se ne possono aggiungere di nuovi, per questo infatti occorre un'altra istruzione, INSERT, che serve ad inserire nuovi record (righe) in una tabella. La tabella deve essere indicata con il nome dopo l'ulteriore parola chiave INTO, mentre per indicare i valori da inserire esistono due sintassi alternative.

La prima alternativa prevede che si fornisca l'elenco dei valori da inserire nella riga come elenco di valori, separati da virgole e compresi fra parentesi tonde, che deve seguire l'ulteriore istruzione VALUES. In questo caso i valori verranno inseriti sequenzialmente nelle colonne della tabella, ed il loro numero deve corrispondere a quello delle colonne. Un esempio di questa sintassi potrebbe essere il comando:

```
INSERT INTO tablename  VALUES ('val1', 'val2', 'val3', 'val4');
```

La seconda alternativa consente di indicare i valori da assegnare alle singole colonne, lasciando anche alcune vuote. In questo caso occorrerà fornire prima di VALUES l'elenco delle colonne,

elencate per nomi separati da virgole e racchiusi in parentesi tonde, e poi far seguire l'elenco dei valori, in numero corrispondente a quello delle colonne, che ad esse verranno rispettivamente assegnati. Un esempio di questa sintassi potrebbe essere il comando:

```
INSERT INTO tablename (column1, column3) VALUES ('val1', 'val3');
```

Un'altra operazione comune, dopo modifica ed inserimento, è quella di cancellazione, che viene eseguita attraverso l'istruzione `DELETE` seguita dall'indicazione della tabella su cui operare preceduta dalla ulteriore parola chiave `FROM`. A questa segue normalmente una ulteriore istruzione `WHERE` per indicare quale righe cancellare, dato che altrimenti verrebbero cancellati tutti i contenuti della tabella, cosa che è raramente quello che si vuole. Di nuovo il comando cancellerà tutte le righe selezionata con `WHERE` per cui può essere opportuno controllare con una `SELECT` preventiva, che ha esattamente la stessa sintassi, cosa si va cancellare. Un esempio potrebbe essere il comando:

```
DELETE FROM tablename WHERE column2='value';
```

Infine, ancorché molto più specialistica, un'ultima istruzione che può risultare utile è `JOIN` che consente di unire i dati di due tabelle usando un campo comune, in sostanza qualcosa di simile a quello che fa il comando `join` visto in sez. 2.2.4. In sostanza con `JOIN` si possono sfruttare le capacità relazionali del database ed fare ricerche unendo i dati che si sono suddivisi sulle varie tabelle usando ad esempio le *foreign key* di cui si è parlato in sez. 4.4.1.

La sintassi prevede che `JOIN` venda usato come estensione di una `SELECT`, indicando come argomento il nome della tabella di cui si vuole effettuare la "*giunta*" seguita dalla ulteriore parola chiave `ON` che indica quali sono i campi che devono corrispondere, da indicare nella notazione `tabA.colA=tabB.colB`. In questo caso la `SELECT` deve indicare, con la stessa notazione `tabN.colN` quali campi delle due tabelle si vogliono ottenere, e verranno restituiti in una unica riga i dati presenti nelle due tabelle per cui le colonne indicate hanno lo stesso valore.

In sostanza se i valori di una colonna di una tabella, in genere un identificativo univoco, sono usati come valori per un'altra colonna in un altra tabella, che contiene altri dati relativi a questo identificativo, si può generare una lista di tutte le possibili corrispondenze. Nell'esempio di autori e libri cui si è accennato in sez. 4.4.1 allora, posto che si sia usato una colonna con un identificativo unico (una *primary key*, diciamo nella colonna `id_autore`) nella tabella degli autori e che si sia usato lo stesso identificativo dell'autore nella tabella dei libri (come *foreign key*, diciamo nella colonna `autore`) si potrà ottenere una lista di tutti i libri e tutti gli autori con un comando del tipo:

```
SELECT autori.nome, autori.cognome, libri.titolo, libri.editore
FROM autori JOIN libri
ON autori.id_autore=libri.autore;
```

In realtà il comando prevede diverse varianti, anche se quella più comune è quella illustrata che viene più precisamente indicata come `INNER JOIN`. Per maggiori dettagli si rimanda ad un qualunque testo elementare che tratti le basi dell'SQL o alla documentazione dei principali server di database.

Capitolo 5

Amministrazione straordinaria del sistema

5.1 La gestione dei dischi e dei filesystem

In questa sezione, dopo una introduzione su alcune nozioni di base relative all'hardware, tratteremo in dettaglio le modalità con cui vengono gestiti in un sistema GNU/Linux dischi e filesystem, quelle con cui si può accedere ai dati in essi contenuti, le strategie di partizionamento e tutto quanto riguarda le operazioni di manutenzione, gestione e controllo degli stessi.

5.1.1 Alcune nozioni generali

Prima di affrontare l'uso dei programmi per la gestione di dischi e filesystem occorre introdurre qualche concetto relativo al funzionamento della gestione dei dischi. In genere, e per questo sono chiamati dispositivi a blocchi, lo spazio disponibile sui piatti magnetici di un disco viene suddiviso in blocchi elementari di dati delle dimensioni di 512 byte chiamati *settori*, anche se con alcuni dischi più recenti e le ultime versioni degli standard è divenuto possibile usare settori di 4096 byte. Dal disco si può sempre leggere un settore alla volta e non un singolo byte. Questo viene fatto direttamente dal kernel o dal BIOS che dice all'interfaccia hardware di leggere un determinato settore fornendogli l'indirizzo dello stesso.

In generale l'accesso ai dischi deve essere eseguito attraverso una apposita interfaccia hardware che consente al processore di farsi inviare in memoria i dati mantenuti sul disco. Inizialmente nel mondo dei PC la stragrande maggioranza dei computer montava come interfaccia per l'accesso ai dispositivi a blocchi (dischi, CDROM) una interfaccia di gestione chiamata alternativamente IDE (acronimo di *Integrated Drive Electronics*, come indicato nella specificazione originaria della Western Digital) o ATA (acromino di *Advanced Technology Attachment*).

Le interfacce IDE/ATA sono state supportate da Linux fin dalle prime versioni del kernel. Per il loro classico utilizzo come interfacce per l'accesso ai dischi, le relative periferiche sono da sempre associate a file di dispositivo che iniziano con il prefisso `hd` (da *hard disk*), con una terza lettera che, in ordine alfabetico, identifica la periferica a cui si fa riferimento.

A lungo l'unica alternativa alle interfacce IDE/ATA è stata l'interfaccia SCSI (vedi sez. 5.4.2) che fin dall'inizio presentava una serie di vantaggi tecnici, come quello di permettere il collegamento di più periferiche su uno stesso cavo e quello di avere velocità di trasferimento più alte, ma lo svantaggio di essere nettamente più costosa.

Dato che con una interfaccia SCSI l'accesso alle periferiche viene gestito in maniera completamente diversa rispetto ad IDE anche i file di dispositivo sono stati indicati in maniera diversa, utilizzando il prefisso `sd` (da *SCSI disk*), sempre seguito da una lettera progressiva in ordine alfabetico che identifica i diversi dischi presenti, assegnata secondo l'ordine di riconoscimento degli stessi.

Questo significa che il primo disco SCSI riconosciuto sarà `/dev/sda`, il secondo `/dev/sdb`, il terzo `/dev/sdc`, ecc. Si tenga presente però che a differenza delle interfacce IDE, il file di dispositivo generico `/dev/sdX` viene usato soltanto per i dischi, vedremo in sez. 5.4.2 che per gli altri tipi di dispositivi (come i CDROM) vengono utilizzati nomi diversi.

Una caratteristica importante dell'interfaccia SCSI è quella che ha visto fin dal principio la creazione di un protocollo generico per gestire l'accesso ai dischi, utilizzabile con qualunque disco SCSI. Questo protocollo è stato implementato nel kernel in maniera indipendente dal supporto per lo specifico hardware sottostante (vedi fig. 5.6) per cui è stato possibile riutilizzarlo anche con interfacce di comunicazione completamente diverse come l'USB (vedi sez. 5.4.4) o la IEEE 1394 (più nota come *FireWire* o *i.LINK*). Questo ha fatto si che anche con queste interfacce si è continuato ad usare per i file di dispositivo gli stessi nomi di una interfaccia SCSI vera e propria.

Inoltre oggi le interfacce IDE/ATA classiche sono ormai universalmente sostituite dallo standard *Serial ATA* (SATA), che consente velocità di comunicazione più elevate con un cablaggio più compatto. In questo caso, a parte un breve periodo di transizione in cui anche questi dispositivi venivano trattati come i normali IDE/ATA, in tutti i kernel attuali l'accesso è gestito in maniera unificata attraverso una libreria interna, chiamata *libATA*, che di nuovo usa il protocollo SCSI per gestire la comunicazione con i dischi.

Infine più recentemente anche il supporto di molti *chipset* che usano il tradizionale standard IDE/ATA (quello che ormai viene chiamato *parallel ATA*) è stato riscritto per utilizzare l'accesso tramite *libATA*. Il risultato finale è che al giorno d'oggi praticamente tutti i dischi vengono visti come dischi SCSI e sono acceduti attraverso file di dispositivo nella forma `/dev/sdX`, pur essendo collegati fisicamente con le interfacce più diverse. Le eccezioni a questa regola ormai sono soltanto alcuni controller RAID hardware (vedi sez. 6.1.2) ed i dischi virtuali ad uso dei sistemi di virtualizzazione.

5.1.2 Il partizionamento dei dischi

Uno dei compiti di preparazione da eseguire tutte le volte che si installa un nuovo disco è quello di suddividerne lo spazio in opportune *partizioni*. In genere questo viene fatto dal programma di installazione del sistema, che in tutte le distribuzioni più recenti è in grado di eseguire il compito in maniera semi-automatica. Quando però si vuole installare un nuovo disco, o si vuole effettuare l'operazione manualmente per avere un maggior controllo, occorre saper utilizzare un programma di partizionamento.

Prima di entrare nel dettaglio del funzionamento dei vari programmi di partizionamento è opportuno fornire alcune informazioni di base relativamente alla gestione delle partizioni. Le partizioni nacquero infatti come funzionalità fornita dal BIOS per consentire di suddividere un

singolo disco in più parti, in modo che queste potessero essere accedute in maniera indipendente l'una dall'altra, come fossero dispositivi separati.

Nell'architettura tradizionale dei PC tutte le informazioni sulle partizioni venivano originariamente mantenute nel primo settore di ciascun disco, il cosiddetto *Master Boot Record* (MBR), chiamato così perché è su di esso che viene installato il *bootloader*, cioè il programma di avvio usato dal BIOS per lanciare il sistema operativo (vedi sez. 5.3). Una parte di questo settore costituisce la cosiddetta *tabella delle partizioni*, dove vengono memorizzati i valori del settore di inizio e di fine di quelle che oggi sono note come *partizioni primarie*. Dato lo spazio limitato disponibile nell'MBR, nella tabella delle partizioni non possono essere memorizzate più di quattro partizioni.

Architetture meno obsolete, come SPARC, PowerPC, o Alpha, o anche i PC più recenti con EFI e GPT (che tratteremo più avanti) non hanno mai avuto questo limite, ed il numero di partizioni è molto più grande, ad esempio sui vecchi MacIntosh PowerPC il limite era di 16. Nell'architettura PC, per mantenere la compatibilità con le vecchie versioni e superare questa limitazione, sono state introdotte le cosiddette *partizioni logiche*. Per poterle utilizzare è necessario marcare una delle partizioni primarie mantenute nella tabella delle partizioni dell'MBR come *partizione estesa* e a questo punto sarà possibile utilizzare il primo settore di detta partizione per installare una tabella secondaria che è quella su cui si indicheranno le ulteriori partizioni logiche, per questo motivo le partizioni logiche talvolta vengono chiamate anche *partizioni secondarie*.

Partizioni primarie e secondarie possono coesistere, nei termini in cui si usa lo spazio di una partizione primaria come partizione estesa dove indicare la tabella delle successive partizioni secondarie. In genere questo si fa creando tre partizioni primarie ed utilizzando la quarta come partizione estesa per poter poi creare anche le partizioni secondarie; ma si può usare una partizione primaria qualunque, (ad esempio la seconda) tenendo presente però che quando si usa una partizione primaria come partizione estesa per introdurre le partizioni logiche, le partizioni primarie successive (nell'esempio la terza e la quarta) non saranno più utilizzabili, anche se in realtà se la partizione estesa non esaurisce tutto lo spazio disco questo non è vero, ma questo tipo di configurazione non è consigliata dato che non tutti i sistemi o i programmi di partizionamento sono in grado di capirla.

Il kernel legge la tabella delle partizioni all'avvio, ed usa per ciascuna partizione un diverso file di dispositivo grazie al quale è possibile accedere direttamente al contenuto della stessa. Questo comporta che se si modifica la tabella delle partizioni il kernel può non essere in grado di vedere le modifiche fino al riavvio successivo, se il disco è in uso infatti la funzione che consente la rilettura della tabella delle partizioni fallirà, per evitare che il kernel possa usare una versione non corrispondente a quella in uso al momento.

Usualmente il nome del file di dispositivo associato ad una partizione viene ottenuto aggiungendo il numero della partizione al nome del file di dispositivo associato al disco. Le partizioni primarie sono numerate da 1 a 4, mentre quelle logiche vanno dal 5 in poi, tradizionalmente il kernel prevede un massimo di 63 partizioni per i dischi IDE e di 16 per gli SCSI, con dei limiti che originano da ragioni storiche relative ai valori dei *minor number* (vedi sez. 1.4.4) dei relativi dispositivi, in realtà oggi, grazie all'uso di LVM (vedi sez. 6.2) l'uso di un gran numero di partizioni non è più necessario, potendo utilizzare alla bisogna un gran numero di volumi logici.

Sui PC il comando utilizzato tradizionalmente per partizionare un disco è `fdisk`, che si trova con lo stesso nome, anche se con sintassi diverse, anche in altri sistemi operativi. Non tratteremo qui le altre architetture, che in genere hanno programmi diversi. Ad esso si sono in seguito ag-

giunti altri programmi, caratterizzati per lo più da una interfaccia più funzionale o più semplice. In particolare alcuni di questi, come `qtparted` o `gparted`, sono dotati di interfaccia grafica e sono in grado di effettuare anche operazioni di ridimensionamento dei filesystem contenuti nelle partizioni (vedi sez. 6.3.2).

Il comando prende come argomento il file di dispositivo del disco su cui si vuole operare, le uniche opzioni che ha senso usare sono `-l` che si limita a stampare la tabella delle partizioni presente e `-s` che stampa le dimensioni totali di una partizione (se passata come argomento con il relativo file di dispositivo) o del disco. Le altre opzioni servono per specificare le caratteristiche del disco e normalmente non servono in quanto i valori sono determinati automaticamente, e per quanto riguarda numero di cilindri, testine e settori possono essere modificati anche all'interno del comando.

Quando il comando viene lanciato stampa un messaggio di benvenuto e, qualora presenti avvisa di potenziali problemi, un esempio del risultato del comando è il seguente:

```
hain:/root# fdisk /dev/sda

WARNING: DOS-compatible mode is deprecated. It's strongly recommended to
         switch off the mode (command 'c') and change display units to
         sectors (command 'u').

Command (m for help):
```

dopo la stampa iniziale `fdisk` si pone in attesa dei comandi che vengono specificati con le relative lettere; se vogliamo una lista dei comandi disponibili possiamo premere `m` come indicato, ottenendo:

```
Command (m for help): m
Command action
   a   toggle a bootable flag
   b   edit bsd disklabel
   c   toggle the dos compatibility flag
   d   delete a partition
   l   list known partition types
   m   print this menu
   n   add a new partition
   o   create a new empty DOS partition table
   p   print the partition table
   q   quit without saving changes
   s   create a new empty Sun disklabel
   t   change a partition's system id
   u   change display/entry units
   v   verify the partition table
   w   write table to disk and exit
   x   extra functionality (experts only)
```

questo ci dice che con `p` possiamo stampare la tabella delle partizioni corrente, nel caso otterremo:

```
Command (m for help): p

Disk /dev/sda: 1798.8 GB, 1798752436224 bytes
255 heads, 63 sectors/track, 218685 cylinders, total 3513188352 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
```

```
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x0004fb07

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048     1953791      975872   83  Linux
/dev/sda2         1953792   783204351   390625280   83  Linux
/dev/sda3       783206398  3513186303  1364989953    5  Extended
/dev/sda5       783206400  3513186303  1364989952   8e  Linux LVM
```

che ci mostra come nel disco siano presenti quattro partizioni.

A questo punto si potrà usare `d` per cancellare una partizione (il comando ne chiederà il numero), ed `n` per crearne una nuova, nel qual caso `fdisk` prima chiederà se primaria o secondaria e poi il relativo numero. Una volta scelta la partizione il programma chiederà il settore di partenza, proponendo come default il primo che risulta libero, e la dimensione. Quest'ultima può essere specificata in varie unità di misura o direttamente con il settore finale, il comando propone sempre come default l'ultimo settore del disco.

Il comando `t` permette di impostare il valore numerico che identifica il *tipo* di partizione; questo viene in genere utilizzato per indicare che tipo di filesystem sarà contenuto nella stessa. Per i filesystem di Linux il valore è `83` (in esadecimale) e vale per tutti i filesystem. I filesystem Windows invece usano vari codici diversi a seconda del tipo di filesystem. Il valore `82` è invece usato per indicare una partizione destinata all'uso come *swap* (vedi sez. 5.1.7). Altri valori usati da Linux sono `fd` o `da` per indicare le partizioni usate per il RAID software (vedi sez. 6.1.3) e `8e` per quelle usate per il *Logical Volume Manager* (vedi sez. 6.2).

Qualunque modifica si sia effettuata sulla tabella delle partizioni questa non verrà scritta su disco a meno di non effettuare una richiesta esplicita di scrittura con il comando `w`; questo comunque chiederà conferma prima di effettuare la scrittura, avvertendo che si possono perdere tutti i dati presenti sul disco. Si esce da `fdisk` con il comando `q`, che non salva le modifiche effettuate.

Un comando alternativo a `fdisk` è `cfdisk`, un programma più *amichevole* che permette l'uso una interfaccia testuale semigrafica ed interattiva in cui si possono eseguire le varie operazioni spostandosi su comandi e partizioni con l'uso delle frecce. In fig. 5.1 è mostrata la schermata iniziale di `cfdisk`, dalla quale è possibile dare tutti i comandi.

Infine un terzo programma che consente di manipolare la tabella delle partizioni è `sfdisk`, che ha la caratteristica di operare a riga di comando in modalità non interattiva, e pertanto può essere utilizzato anche all'interno di script. Il comando prende sempre come argomento il file di dispositivo relativo ad un disco (ad esempio `/dev/sda`), tranne che quando invocato con l'opzione `-s`, che stampa la dimensione delle partizioni, nel qual caso si può usare anche il file di dispositivo di una singola partizione. Con l'opzione `-l` invece il comando stampa la tabella delle partizioni in un formato leggibile, e se non si specifica nessun dispositivo esegue l'operazione per tutti i dischi.

Se invocato senza opzioni il comando legge dallo standard input le istruzioni relative al partizionamento del disco indicato; quando invocato su un terminale le istruzioni sono richieste in modalità interattiva, ma qualora si sia rediretto lo standard input queste possono essere lette direttamente da un file consentendo così di automatizzare le operazioni di partizionamento in modalità non interattiva.

Le istruzioni consentono di cancellare, spostare e modificare le partizioni, e seguono una sintassi specifica (per i dettagli si può consultare la pagina di manuale); benché il comando

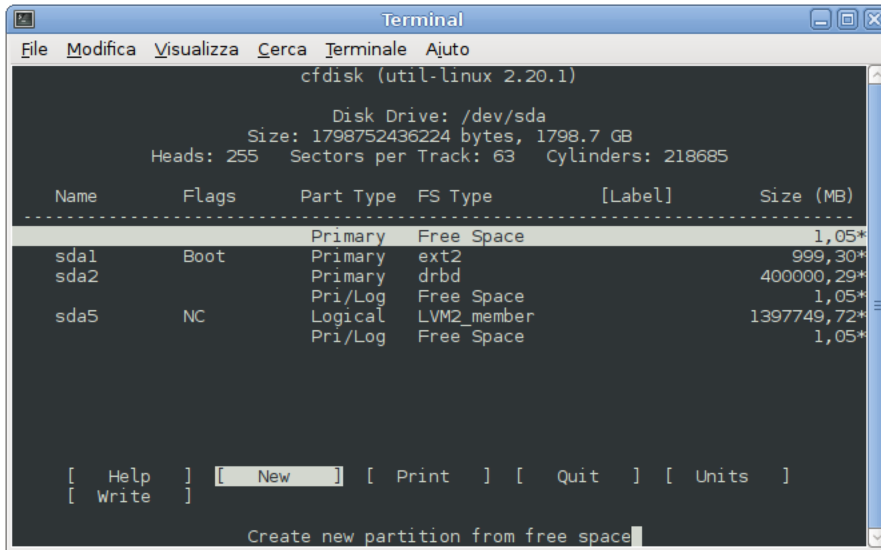


Figura 5.1: Schermata del comando `cfdisk`.

abortisca qualora si commetta un errore di sintassi, si deve avere ben presente che basta inserire un qualche valore sbagliato per perdere il partizionamento di un disco. Per questo vengono fornite le opzioni `-O` e `-I`.

L'opzione `-O` consente di salvare, sul file indicato come parametro, il contenuto di tutti i settori modificati dalla conseguente esecuzione di `sfdisk`. Questo significa che se si devono effettuare delle modifiche con `sfdisk` è sempre opportuno invocarlo come `sfdisk /dev/sda -O sda.save`, in questo modo se si è commesso un errore sarà possibile ripristinare la situazione precedente invocando il comando con l'opzione `-I`, sarà cioè possibile tornare indietro con `sfdisk /dev/sda -I sda.save`.

In realtà l'uso di `sfdisk` per eseguire delle modifiche manuali è piuttosto ridotto, dato che un comando come `cfdisk` è molto più comodo; dove esso diventa molto utile è quando si vogliono partizionare molti dischi nella stessa maniera come nel caso in cui si devono impostare le partizioni dei vari dischi componenti un RAID 1 (vedi sez. 6.1). In tal caso infatti è possibile eseguire il partizionamento del primo disco, e poi salvare la struttura della tabella delle partizioni con l'opzione `-d`, che genera sullo standard input una serie di istruzioni che possano essere a loro volta utilizzate come input per lo stesso `sfdisk`; così se si vuole replicare il partizionamento di `/dev/sda` su `/dev/sdc` basterà eseguire il comando:

```
sfdisk -d /dev/sda | sfdisk /dev/sdc
```

Si tenga presente però che `-d` non è una alternativa all'uso di `-O` per ripristinare un cambiamento di tabella delle partizioni, se infatti si può riportare la tabella delle partizioni nella forma originaria, non saranno recuperate eventuali modifiche fatte ad altre parti del disco, come può accadere qualora si creino delle partizioni logiche su dischi ove non esistevano e poi si voglia tornate indietro.

Opzione	Significato
`-s`	stampa la dimensione delle partizioni di un disco o della singola partizione.
`-l`	stampa una lista delle partizioni presenti sul disco.
`-d`	scrive il contenuto della tabella delle partizioni di un disco nel formato delle istruzioni di `sfdisk`, così che l'output del comando può essere usato come input per una successiva invocazione dello stesso.
`-V`	esegue una verifica della tabella delle partizioni e riporta ogni eventuale anomalia.
`-n`	simula le operazioni senza scrivere nulla sul disco.
`-O` *file*	salva sul file passato come parametro il contenuto binario dei settori modificati dal comando.
`-I` *file*	ripristina il contenuto del disco utilizzando le informazioni ottenute dal file passato come parametro (che deve essere stato prodotto usando l'opzione `-O` nella precedente invocazione del comando).

Tabella 5.1: Le principali opzioni del comando `sfdisk`.

Come accennato all'inizio la tabella delle partizioni mantenuta nell'MBR comporta molte limitazioni. Oltre al numero delle stesse la più seria è che essendo il numero di settore che le delimita indicato con valori a 32 bit, la dimensione massima utilizzabile per un disco, con settori di 512 byte, e di due terabyte, ormai già superata da vari produttori. Per questo, anche se viene comunque supportata per compatibilità, nelle macchine più recenti, insieme all'uso di UEFI (vedi sez. 5.3) al posto del BIOS tradizionale, è stato adottato un nuovo formato delle partizioni denominato GPT (*GUID Partition Table*).

Il nuovo formato prevede che vengano utilizzati i primi 34 e gli ultimi 33 settori del disco. Il primo settore è riservato e non viene mai utilizzato da GPT. Questo consente di mantenervi un MBR tradizionale contenente una tabella delle partizioni classica che può essere utilizzato su macchine che non supportano GPT, oppure di proteggere la partizione GPT stessa allocando tutto lo spazio disponibile in modo che i programmi che non la riconoscono evitino di sovrascriverla.

Il settore seguente (che viene replicato sull'ultimo settore del disco) contiene la cosiddetta *intestazione primaria* (*Primary GPT header*) con un GUID (un *Globally Unique Identifier*, un identificativo pseudocasuale di 16 byte) che identifica il disco, la posizione dell'*intestazione secondaria* (la *Secondary GPT header* che viene copiata sull'ultimo settore del disco), il numero e le dimensioni degli elementi mantenuti della successiva tabella della partizioni, ed una *checksum* globale che viene usata dal *firmware* di avvio (UEFI, vedi sez. 5.3.1) per verificarne l'integrità.

I dati della tabella vengono poi memorizzati nei seguenti 32 settori successivi e prevedono per ciascuna partizione un altro GUID che ne identifica il tipo (che fa le veci dell'analogo di un solo byte nell'MBR) seguiti dall'inizio e la fine della stessa in termine di numero di settori (indicato a 64 bit) e dall'eventuale nome ed attributi della stessa. Questi 32 settori vengono copiati per ridondanza anche in fondo al disco, immediatamente sotto l'intestazione secondaria.

Dato che il formato è completamente diverso, per la gestione delle partizioni GPT sono stati creati altri programmi analoghi ai precedenti, caratterizzati dall'uso della "`g`" al posto della "`f`". Pertanto il comando di base per modificare una partizione GPT è `gdisk`, che ha un comportamento analogo a `fdisk` (anche per il significato dell'opzione `-l`); se invocato su un disco

partizionato con GPT otterremo infatti:

```
hain:/root# gdisk /dev/sda
 GPT fdisk (gdisk) version 0.8.1

Partition table scan:
  MBR: protective
  BSD: not present
  APM: not present
  GPT: present

Found valid GPT with protective MBR; using GPT.

Command (? for help):
```

I comandi, ottenibili stavolta premendo "`?`", sono analoghi a quelli di `fdisk`, ad esempio premendo "`p`" si può stampare la tabella delle partizioni, ottenendo qualcosa come:

```
Command (? for help): p
Disk /dev/sda: 5860533168 sectors, 2.7 TiB
Logical sector size: 512 bytes
Disk identifier (GUID): 5FDEE2EA-D7CE-4007-BD9C-5BD47FCF6273
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 5860533134
Partitions will be aligned on 2048-sector boundaries
Total free space is 2014 sectors (1007.0 KiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1            4096         8392703   4.0 GiB     FD00
   2         8392704        39849983   15.0 GiB    FD00
   3        39849984      5860533134   2.7 TiB     FD00
   4            2048            4095   1024.0 KiB  EF02
```

Anche i restanti comandi principali sono analoghi a quelli di `fdisk`; con "`d`" si può cancellare una partizione indicandone poi il numero, con "`n`" crearne una nuova, di nuovo si dovrà indicare il numero e poi inizio, fine e tipo. Con "`q`" si può uscire dal programma e come per `fdisk` nessuna modifica verrà registrata se non avendo fatto una richiesta esplicita con "`w`" confermando la scrittura. Inoltre è possibile salvare un backup della tabella delle partizioni con "`b`" (chiederà il nome del file su cui scriverlo) e ricaricarlo con "`l`".

La differenza più significativa resta nell'indicazione del tipo di partizione, che si può fare su una partizione esistente usando "`t`" ed indicando poi il numero della stessa, in questo caso visto la complessità di scrivere un GUID di 16 byte, vengono usati valori di due byte (quattro cifre esadecimali) analoghi a quelli usati per `fdisk` cui vengono aggiunti due zeri, la cui lista si ottiene premendo `L`; la conversione al corretto GUID viene realizzata internamente da `gdisk`.

Oltre a `gdisk` esistono gli ulteriori comandi `cgdisk`, che replica l'interfaccia semigrafica di `cfdisk`, e `sgdisk` che consente di eseguire le operazioni in modalità non interattiva come `sfdisk` anche se in questo caso la sintassi del comando è completamente diversa. L'uso di quest'ultimo comando è comunque sconsigliato se non per l'uso all'interno di script, in quanto a differenza di `sfdisk` non effettua nessun controllo di errore sulle informazioni che gli vengono passate, ed è pertanto facile fare danni.

Ovviamente il partizionamento di un disco è una operazione privilegiata, che può compiere solo l'amministratore, e richiede il permesso di scrittura sul dispositivo su cui si esegue il relativo

comando. E come accennato è anche un compito molto delicato, in quanto se si cambia la tabella delle partizioni cancellando o modificando una di quelle presenti il relativo contenuto andrà perso. Occorre quindi stare molto attenti ed evitare di salvare le modifiche fintanto che non si è assolutamente sicuri di quanto si è fatto.

Infatti, anche se in realtà gli unici dati che vengono cambiati sono quelli della tabella delle partizioni, una volta che questa è stata modificata non saranno più presenti sul disco i confini delle partizioni preesistenti e dato che ogni filesystem viene creato per stare esattamente nelle dimensioni di una partizione, se se ne modificano dimensioni o posizione i dati del filesystem in essa contenuto non saranno più utilizzabili.

Questo ci dice anche che, fintanto che non si va a modificare il contenuto effettivo del disco, l'accesso ai dati può essere riottenuto ripristinando la tabella delle partizioni. Per farlo esistono molti metodi, a partire da quello più semplice di tutti che consiste nel segnarsi il contenuto precedente, operazione che come abbiamo visto è semplicissima da fare con `sfdisk` e con `gdisk`.

Un altro metodo, più complesso, ed utilizzabile solo se nel frattempo non si è riavviata la macchina, è quello di recuperare i dati dalle informazioni mantenute dal kernel attraverso il contenuto di `/proc/partitions` ed i risultati del comando `hdparm`. Si ricordi infatti che il kernel legge la tabella delle partizioni all'avvio e la mantiene in memoria e che è per questo che se si ripartiziona un disco in uso occorre riavviare il sistema per vedere le modifiche.

Infine si può usare il comando `gpart`, da eseguire a disco non montato, che è in grado di trovare, utilizzando opportuni criteri di identificazione dei dati dei filesystem, i confini di partizioni preesistenti, così da poterle ripristinare. In questo caso ovviamente i tempi sono molto più lunghi, ed i risultati non sono sicuri.

Il fatto che la modifica delle dimensioni di una partizione comporti, a meno di opportune contromisure di salvaguardia, la perdita dei dati in essa contenuti ci fa capire che è sempre opportuno pianificare con cura le dimensioni assegnate alle varie partizioni fin dall'installazione del sistema, perché modificarle in un secondo tempo comporta spesso la necessità di dover fare un backup dei contenuti. Infatti, a parte il caso in cui la si allarga con dello spazio disponibile alla sua fine, in genere nel modificare una partizione occorre anche operare sul filesystem in essa contenuto, e questo può comportare diversi problemi. Vedremo in sez. 6.3.2 il comando `parted` che permette di ridimensionare un filesystem e restringere, allargare o spostare la partizione che lo contiene, ma l'operazione resta complessa e delicata.

Tutto questo significa che nell'installazione occorre decidere una buona *strategia di partizionamento*, cioè decidere quante partizioni fare e come assegnarle. Per questo occorre tenere ben presenti le indicazioni sul contenuto delle varie directory di sistema illustrate in sez. 1.2.3. Una delle strategie più semplici è quella di creare una partizione unica e mettere tutto quanto su di essa. Questo schema ha il vantaggio di non doversi preoccupare del dimensionamento delle altre partizioni né del rischio di esaurire lo spazio su una partizione mentre le altre sono vuote. Ha però lo svantaggio che qualora si debba eseguire una reinstallazione verranno sovrascritti i dati degli utenti (normalmente quelli sotto `/home`) e quelli di sistema (come pagine web, posta archiviata e tutto quanto in genere si tiene sotto `/var` o `/srv`). Inoltre se si hanno più dischi non è ovviamente possibile mettere tutto su una sola partizione. Infine anche qualora si usi una sola partizione per tutto il sistema, è in generale una buona pratica creare comunque una seconda partizione per l'area di *swap* (vedi sez. 5.1.7).

Per questo motivo di solito si preferisce creare almeno un'altra partizione su cui montare `/home` per i dati degli utenti. O qualora si abbiano dati di sistema un'altra per `/var` o `/srv`. Se

poi si vuole separare anche quanto non strettamente necessario all'avvio dal resto del sistema si può mettere anche `/usr` su una partizione separata, anche se questo è sconsigliato nei sistemi più recenti, in quanto alcuni programmi si aspettano, nonostante non sia garantito dal FHS, che questa sia posta direttamente sotto la radice.

Inoltre, onde evitare che gli utenti possano riempire la radice con file temporanei, può essere opportuno usare una partizione separata per `/tmp`. Su questo aiuta la quota di disco riservata per l'amministratore che molti filesystem supportano (vedi sez. 5.1.4), salvo il caso in cui è l'amministratore stesso a creare, per sbaglio, troppi file temporanei. Infine in certi casi può essere necessario creare una partizione separata per `/boot`, in particolare quando si esegue una installazione su dispositivi cifrati, in tal caso infatti il *bootloader* dovrà accedere ai file necessari all'avvio (vedi sez. 5.3) in chiaro mentre tutto il resto sarà cifrato.

Usare diverse partizioni (e più dischi) ha comunque una serie di controindicazioni. Anzitutto si può sprecare inutilmente spazio disco quando una partizione si riempie mentre le altre sono vuote, e a questo punto si ha solo l'alternativa di ripartizionare o di spostare pezzi da una partizione all'altra e farvi riferimento con dei link simbolici, che non è il massimo della semplicità di gestione e può comportare una serie di problemi con i programmi di backup che di norma, per evitare di inserire nel backup cose che non c'entrano nulla, prevedono di archiviare solo i file presenti sul filesystem corrente.

Inoltre una volta che si sono spostate su partizioni diverse `/home`, `/var`, `/srv`, `/tmp` e `/usr` (ed eventualmente `/opt`) non è che resti molto altro da spostare, a parte esercizi di stile come lo spostamento di `/var/tmp` o `/usr/local`, la cui utilità, visto l'aumento della complessità di gestione, è molto dubbia. Inoltre per quanto grandi si possano fare le partizioni, non si risolverà mai il problema di esaurire lo spazio in quelle con la maggior parte dei contenuti, quando l'occupazione dei dati di una partizione cresce oltre la capacità del più grande dei dischi che si hanno.

In casi come questo si è costretti ad usare più dischi dividendo i contenuti che normalmente starebbero sotto una sola directory, cosa che può rendere la gestione molto più complicata, specie se questo accade per directory che non siano `/home`. Per quest'ultima infatti, che è quella più facilmente soggetta alla possibilità di esaurimento dello spazio, è sempre stata prevista, sia pure al prezzo di una maggiore attenzione nella creazione degli utenti (si ricordi quanto visto in sez. 4.3.3), la possibilità di suddividerne il contenuto su più dischi, creando le home directory degli utenti su pathname diversi.

Come si vede l'uso delle partizioni comporta una serie di potenziali problemi di gestione, dovuti alla necessità di pianificare adeguatamente in fase di installazione la gestione dello spazio disco, cosa che non sempre è possibile fare in maniera sicura, anche perché le condizioni di uso di una macchina possono cambiare. Per risolverli alla radice è perciò disponibile, a partire dai kernel della serie 2.4.x, il sistema del *Logical Volume Manager* che permette di unire partizioni[1] e dischi fisici diversi, utilizzando poi dei *volumi logici* all'interno dei quali creare i relativi filesystem senza dover fare riferimento diretto ad una specifica partizione. Tratteremo questo argomento in sez. 6.2.

[1] si noti come in fig. 5.1 compaia appunto una partizione di tipo `Linux LVM`, che è quella da usare per i volumi fisici che verranno usati con LVM come componenti di un volume logico.

5.1.3 La gestione del contenuto dei dischi

Come illustrato in sez. 1.2.3 una delle caratteristiche dei sistemi unix-like che disorientano maggiormente chi proviene da altri sistemi operativi è la presenza di un unico albero delle directory. Dato che il concetto di volume o disco come entità separata non esiste, la conseguenza è che il contenuto dei dischi, o degli altri dispositivi contenenti file come CDROM, floppy o chiavette USB, deve essere fatto apparire all'interno dell'unico albero dei file, così che possa essere visto dal sistema all'interno di opportune directory.

Come già accennato l'operazione che permette di fare comparire i dati del filesystem contenuto in un dispositivo all'interno di una directory viene chiamata "*montaggio*", ed una delle caratteristiche di Linux (e degli altri Unix) è che all'avvio del sistema il kernel si limita a montare soltanto la directory radice che deve il suo nome appunto all'essere la *radice* dell'albero dei file.[2] Questo significa che se si è seguita la prassi comune di installare i dati su partizioni separate, queste dovranno essere montate in un secondo momento. Lo stesso dicasi se si vogliono utilizzare dati presenti in un dispositivo (floppy, CDROM, o altro) reso disponibile in un tempo successivo all'avvio.

Le distribuzioni moderne vengono incontro all'utente prevedendo, come supporto per il software installato insieme all'interfaccia grafica, un meccanismo di riconoscimento automatico di eventuali altri "*volumi*" resisi disponibili, con tanto di montaggio automatico degli stessi, ciò non toglie che l'operazione debba comunque essere effettuata. Noi vedremo qui come lo si fa in maniera "*manuale*", che è la modalità con cui lo si fa normalmente sui server dove l'interfaccia grafica in genere non viene neanche installata.

Il comando che permette di montare il filesystem contenuto in un disco all'interno di una directory è `mount`, che di norma si limita ad invocare la omonima *system call* del kernel. Si tenga presente poi che per alcuni filesystem (in particolare per quelli di rete come `nfs` e `smbfs`) per l'esecuzione del comando non è sufficiente la chiamata alla omonima *system call*, ma devono essere usati dei programmi ausiliari; questi vengono lanciati con l'invocazione automatica di un corrispondente programma `/sbin/mount.fs_type`.

Se lo si invoca senza argomenti il comando si limita a mostrare l'elenco dei filesystem attualmente montati, ma normalmente esso viene sempre invocato nella forma:

```
mount -t fs_type /dev/device /path/to/dir
```

dove l'opzione `-t` serve a specificare il tipo di filesystem che è contenuto nel dispositivo che si vuole montare, che deve essere indicato tramite il suo file di dispositivo `/dev/device`. Infine `/path/to/dir` indica la directory, che viene detta *mount point*, in cui il filesystem verrà *montato*, vale a dire la directory all'interno della quale comparirà il contenuto del filesystem.

Il comando richiede quindi la conoscenza del tipo di filesystem presente nel dispositivo che si vuole montare, da indicare come parametro per l'opzione `-t`. L'elenco dei nomi dei principali tipi di filesystem supportati è riportato in tab. 5.2, è possibile comunque usare anche un meccanismo di ricerca automatico, che viene attivato o usando `auto` come tipo di filesystem o non specificando l'opzione. In questo caso viene effettuato automaticamente un controllo se nel dispositivo è presente uno dei filesystem riconosciuti per il quale viene usato il comando `blkid` che vedremo a breve.

[2] vedremo in sez. 5.3 come definire quale sia il dispositivo su cui si trova il filesystem che contiene la radice sia una delle impostazioni fondamentali relative all'avvio del sistema.

Tipo	Descrizione
adfs	*Acorn Disc Filing System*, il filesystem del sistema operativo RiscOS.
btrfs	*B-tree File System*, un filesystem di nuova generazione, progettato per le funzionalità più avanzate e indicato come futuro sostituto dei vari *extN*.
cramfs	*Compressed ROM File System*, un filesystem su ROM per sistemi embedded.
ext2	*Second Extended File System*, il vecchio filesystem standard di Linux.
ext3	*Third Extended File System*, evoluzione di *ext2* con supporto per il *journaling*.
ext4	*Fourth Extended File System*, ultima versione del filesystem standard di Linux, ulteriore evoluzione del precedente *ext3*.
hfs	*Hierarchy File System*, il filesystem del MacOS (non MacOS X).
hfsplus	*Hierarchy File System Plus*, il filesystem del MacOS X.
hpfs	*High Performance File System*, il filesystem di OS/2.
iso9660	il filesystem dei CD-ROM, definito dallo standard ISO 9660.
udf	*Universal Disk Format*, filesystem per i dispositivi ottici (CD e DVD), successore di ISO 9660.
jfs	*Journaling File System*, il filesystem della IBM con il supporto per il *journaling* portato su Linux.
minix	*Minix File System*, il filesystem del sistema operativo Minix.
ntfs	*NT File System*, il filesystem di Windows NT.
qnx4	*QNX4 File System*, il filesystem usato da QNX4 e QNX6.
reiserfs	*Reiser File System* il primo filesystem di Linux a supportare il *journaling*.
romfs	un filesystem speciale per accedere ai dati memorizzati su una ROM.
ufs	*Unix File System*, il filesystem usato da vari Unix derivati da BSD (SunOS, FreeBSD, NetBSD, OpenBSD e MacOS X).
vxfs	*Veritas VxFS File System*, filesystem standard di UnixWare disponibile anche su HP-UX e Solaris.
xfs	*X File System*, il filesystem di IRIX, portato dalla SGI su Linux.
befs	*BeOS File System*, il filesystem del sistema operativo BeOS.
msdos	il filesystem elementare usato dall'MSDOS.
vfat	il filesystem FAT usato da Windows 95/98.
proc	filesystem virtuale che fornisce informazioni sul sistema.
tmpfs	filesystem usato per mantenere su un'area di memoria dati temporanei che si appoggia al sistema della memoria virtuale, i file non vengono mai salvati su disco, ma in caso di necessità viene utilizzata la partizione di *swap* (vedi sez. 5.1.7) per non perdere il contenuto.
sysfs	filesystem virtuale, introdotto con i kernel della serie 2.6.x, che permette di accedere alle informazioni relative ai dispositivi presenti e relativi driver.
devpts	filesystem virtuale per consentire un accesso efficiente ai terminali virtuali.
nfs	*Network File System*, filesystem per la condivisione di file attraverso la rete tramite il protocollo NFS (vedi sez. 8.4.1) creato da Sun.
coda	*Coda File System*, filesystem distribuito su rete che supporta funzionalità evolute come autenticazione, replicazione e operazioni disconnesse.
smbfs	*SMB File System*, filesystem usato per montare le directory condivise di Windows.

Tabella 5.2: Principali filesystem disponibili su Linux e relativi nomi per l'opzione `-t` di `mount`.

Se il riconoscimento non riesce viene effettuato un ulteriore tentativo tentando di usare i tipi eventualmente elencati in `/etc/filesystems` e poi, se questo file non esiste (come di default) quelli presenti in `/proc/filesystems`, che contiene l'elenco dei filesystem supportati nel kernel in esecuzione. In genere si usa `/etc/filesystems` se si vuole cambiare l'ordine in cui vengono provati i filesystem supportati, ad esempio per provare prima `vfat` di `msdos`, evitando di usare quest'ultimo quando è disponibile il primo, o se si vuole che eventuali moduli aggiuntivi per i

filesystem indicati vengano caricati automaticamente.[3] Se si è creato `/etc/filesystems` si può inoltre richiedere l'uso ulteriore dei dati di `/proc/filesystems` terminandone il contenuto con un asterisco.

Come accennato `/proc/filesystems` contiene l'elenco di tutti i filesystem supportati dal kernel in esecuzione, ed ovviamente per poterlo utilizzare occorre che a sua volta il filesystem virtuale `/proc` (che abbiamo già incontrato in sez. 1.2.3) sia stato preventivamente montato. Il file elenca tutti i filesystem, compresi i cosiddetti *filesystem virtuali*, che non fanno riferimento ai dati di un dispositivo (come lo stesso `/proc`) caratterizzati dalla presenza della parola chiave `nodev`. Ovviamente nel tentativo di riconoscimento automatico di un filesystem questi ultimi non vengono presi in considerazione. Un estratto del contenuto di questo file, preso da una Debian Squeeze, è il seguente:

/proc/filesystems
```
nodev   sysfs
nodev   rootfs
nodev   bdev
nodev   proc
...
        ext3
nodev        rpc_pipefs
nodev        nfs
...
```

Altre due importanti opzioni di `mount` sono `-L` e `-U` che consentono di indicare il filesystem che deve essere montato senza fare riferimento al relativo file di dispositivo. Entrambe necessitano dell'uso del file `/proc/partitions`, dove sono mantenute le informazioni relative alle partizioni dei dischi ed al loro contenuto.

Usando `-L` è possibile indicare quale filesystem montare invece che attraverso il corrispondente file di dispositivo tramite una etichetta (che deve essere stata impostata in fase di creazione del filesystem, vedi sez. 5.1.4), mentre con `-U` si può fare la stessa operazione usando invece il valore del cosiddetto UUID (*Universally Unique Identifier*) associato al filesystem. L'UUID è un numero di 128 bit utilizzato per identificare univocamente un oggetto molto usato nella gestione dei dischi e che ritroveremo in varie occasioni; in genere viene espresso con 32 cifre esadecimali suddivise in 5 sezioni separate dal carattere "`-`"; la prima versione cercava di garantire l'univocità facendo riferimento al *MAC address* (vedi sez. 7.6.4) del proprio computer e ad una marca temporale, le versioni più recenti prevedono una opportuna generazione di un numero univoco.[4]

Queste due opzioni sono oggi utilizzate sempre più spesso, proprio in quanto consentono (in particolare la seconda, che assicura la non ambiguità della identificazione) di rendere non più necessari i riferimenti espliciti ai vari file di dispositivo, che, come vedremo in sez. 5.4.5, ormai vengono allocati prevalentemente in maniera dinamica tramite il sistema di *udev* e possono, in particolare con la diffusione dei dischi rimuovibili, essere di volta in volta diversi.

L'uso di identificativi generici che permettono di riconoscere esattamente un filesystem, indipendentemente dal nome assegnato al rispettivo file di dispositivo, comporta anche la necessità di disporre degli opportuni programmi che consentano di rilevare dette corrispondenze. Se sono note o l'etichetta o l'UUID associato ad un filesystem il rispettivo file di dispositivo può essere

[3] si danno per noti i concetti relativi alla gestione dei moduli del kernel che verranno tratti in sez. 5.2.5.

[4] per i dettagli si consulti `http://en.wikipedia.org/wiki/UUID`.

trovato con il comando `findfs`. Il comando è molto semplice e prende un argomento nella forma `LABEL=etichetta` o `UUID=valore`, stampando il corrispondente file di dispositivo. L'operazione inversa può essere fatta con il comando `blkid`, che prende come argomento un file di dispositivo e stampa come risultato le proprietà del filesystem in esso contenuto, ad esempio:

```
oppish:~# blkid /dev/sda1
/dev/sda1: UUID="aaf4126a-0dc0-4bb8-8429-557eb8c56fba" TYPE="ext3"
```

Il comando stampa le proprietà (file di dispositivo, tipo di filesystem, eventuale etichetta e UUID) relative a qualunque dispositivo a blocchi. Se invocato senza argomenti esegue l'elenco per tutti i dispositivi a blocchi presenti, altrimenti si può effettuare una selezione passando come argomento il nome di un file di dispositivo. Inoltre si può, attraverso l'opzione `-t`, eseguire la selezione per etichetta, UUID o tipo di filesystem, passando all'opzione un parametro rispettivamente nella forma `LABEL=value`, `UUID=value` o `TYPE=value`, dove in quest'ultimo caso il tipo di filesystem deve essere indicato con uno dei valori di tab. 5.2.

Altre opzioni importanti del comando `mount`, utilizzate spesso nell'invocazione, sono `-v`, che aumenta la prolissità dei messaggi e `-a` che monta tutti i filesystem elencati per il montaggio automatico in `/etc/fstab` (sul cui uso torneremo a breve). Si sono riportate le altre principali opzioni in tab. 5.3, mentre per l'elenco completo si può consultare la pagina di manuale, accessibile con `man mount`.

Opzione	**Significato**
`-a`	monta tutti i filesystem marcati per il montaggio automatico all'avvio in `/etc/fstab`.
`-n`	non aggiorna l'elenco dei filesystem montati mantenuto in `/etc/mtab`.
`-o` *opt*	specifica una opzione di montaggio *opt*.
`-r`	monta il filesystem in sola lettura.
`-t` *type*	specifica il tipo di filesystem *type*.
`-w`	monta il filesystem in sola scrittura.
`-v`	aumenta la prolissità nella stampa dei messaggi.

Tabella 5.3: Le principali opzioni del comando `mount`.

Si è lasciata per ultima, da trattare a parte per la sua rilevanza peculiare, l'opzione `-o` che viene usata per specificare le cosiddette *opzioni di montaggio* con sui si possono impostare la caratteristiche di utilizzo del filesystem che viene montato. Ogni filesystem infatti può presentare diverse modalità di funzionamento, che vengono controllate appunto attraverso le sue opzioni di montaggio. Alle opzioni specifiche di ogni singolo filesystem si aggiungono quelle generiche, che sono disponibili per qualunque filesystem, riportate in tab. 5.4.

Ciascuna opzione di montaggio è identificata da un nome, che viene passato come parametro di `-o`; se ne può anche specificare una lista, scrivendone i rispettivi nomi separati da virgole e senza spazi in mezzo. Si tenga presente che alcune delle opzioni elencate in tab. 5.4, in particolare `auto`, `user`, `users` e `defaults` e relative negazioni, hanno significato solo quando vengono usate nel quarto campo del file `/etc/fstab`, su cui torneremo fra breve.

Oltre a specificare delle opzioni di montaggio coi valori riportati in tab. 5.4 l'opzione `-o` consente anche di effettuare alcune operazioni speciali; ad esempio utilizzando il valore `remount` diventa possibile rimontare al volo un filesystem già montato senza smontarlo, per cambiare

Valore	Significato
`async`	tutto l'I/O sul filesystem viene eseguito in maniera asincrona, cioè le funzioni di scrittura ritornano ed è il kernel che si incarica di eseguire la effettiva scrittura dei dati nel momento più opportuno (è il default).
`atime`	Aggiorna il valore del tempo di ultimo accesso ai file (vedi sez. 1.2.2) presenti sul filesystem, era il default fino al kernel 2.6.30.
`auto`	tutti i filesystem con questa opzione citati in `/etc/fstab` vengono montati dal comando `mount -a` (che in genere è quello che viene eseguito all'avvio del sistema).
`defaults`	usa le opzioni di default: `rw`, `suid`, `dev`, `exec`, `auto`, `nouser`, e `async`.
`dev`	consente la presenza di file di dispositivo all'interno del filesystem (è il default).
`dirsync`	esegue in maniera sincrona le operazioni che comportano una scrittura sulle directory: creazione di link, creazione, spostamento e cancellazione di file, creazione e cancellazione di directory e file di dispositivo.
`group`	consente ad un utente di montare il filesystem se questi appartiene al gruppo proprietario del relativo dispositivo; questa opzione comporta automaticamente come restrizioni le opzioni `noexec`, `nosuid`, e `nodev`, se non soprassedute esplicitamente con `exec`, `suid`, e `dev`.
`exec`	consente l'esecuzione di programmi presenti sul filesystem (è il default).
`noatime`	non aggiorna il valore del tempo di ultimo accesso al file (utile quando si vogliono evitare ulteriori accessi al disco).
`noauto`	il filesystem deve essere montato esplicitamente (viene ignorato dall'opzione `-a`).
`nodev`	non consente l'uso di file di dispositivo presenti sul filesystem.
`noexec`	non consente l'esecuzione di programmi presenti sul filesystem.
`nosuid`	non consente che i bit *suid* e *sgid* (vedi sez. 1.4.2) abbiano effetto.
`relatime`	aggiorna il tempo di ultimo accesso al file soltanto quando questo è anteriore al tempo di ultima modifica col le stesse prestazioni di `noatime` senza causare ai programmi che usano il tempo di ultimo accesso per identificare i file che sono stati letti. A partire dal kernel 2.6.30 questa opzione è diventata il default, e può riottenere il precedente comportamento con l'indicazione esplicita della nuova opzione `strictatime`.
`ro`	chiede il montaggio del filesystem in sola lettura.
`rw`	chiede il montaggio del filesystem in lettura e scrittura (è il default).
`suid`	consente che i bit *suid* e *sgid* (vedi sez. 1.4.2) abbiano effetto (è il default).
`sync`	tutto l'I/O sul filesystem deve essere sincrono (vale a dire che le funzioni di scrittura prima di proseguire aspettano che i dati vengano scritti su disco).
`user`	consente anche ad un utente normale di montare il filesystem, con le stesse restrizioni viste per `group`; il nome utente viene scritto su `/etc/mtab` e solo lui potrà smontarlo.
`nouser`	solo l'amministratore può montare il filesystem (è il default).
`users`	consente ad un qualunque utente di montare il filesystem, con le stesse restrizioni viste per `group`, e ad un qualunque altro di smontarlo.

Tabella 5.4: Valori delle opzioni di montaggio disponibili per qualunque tipo di filesystem.

alcune delle opzioni precedenti. Uno degli usi più comuni per questa opzione è quello di rimontare in lettura/scrittura un filesystem che si è montato in sola lettura per poterci effettuare un controllo, o viceversa.[5]

Un'altra opzione molto utile è `loop`; in questo caso in realtà non si tratta propriamente di

[5] si tenga presente però che fin quando ci sono file aperti in lettura/scrittura, l'operazione di rimontaggio in sola lettura fallirà.

una opzione di montaggio relativa ad un filesystem quanto di un modo per effettuare automaticamente l'impostazione di un dispositivo di *loopback* tramite il quale diventa possibile montare un filesystem contenuto in un ordinario file di dati (ovviamente il file deve contenere un filesystem completo). Così se ad esempio il file `geomorphix.iso` è l'immagine ISO di un CD, si potrà accedere al contenuto della stessa in maniera trasparente, montando il file come se fosse un CD, con il comando:

```
mount -t iso9660 -o loop geomorphix.iso /mnt/images
```

Con questa stessa interfaccia è possibile anche montare un filesystem opportunamente cifrato, nel qual caso si dovrà specificare l'opzione `encryption` per indicare l'algoritmo di cifratura usato e l'opzione `keybits` per specificare la lunghezza (in bit) della chiave; la password di accesso verrà chiesta sul terminale. Questo uso è comunque deprecato, e non più disponibile sui kernel recenti, in quanto presenta grossi problemi di sicurezza, al suo posto può essere utilizzato il sistema del *device mapper* che vedremo in sez. 6.3.3.

Come accennato l'uso dell'opzione `loop` è un meccanismo semplificato per eseguire automaticamente l'impostazione di un dispositivo di *loopback*. Questo non è altro che un dispositivo virtuale che consente di accedere al contenuto di un file di dati come se si trattasse di un dispositivo fisico. In generale l'impostazione deve essere fatta manualmente con il comando `losetup`, che consente di associare il contenuto di un file, passato come argomento, ad un apposito dispositivo, accessibile con il nome `/dev/loopN`, dove `N` è un numero progressivo che, se non si specifica niente, viene allocato automaticamente.[6]

Il comando consente anche, con `-f`, di indicare quale specifico file di dispositivo deve essere usato, passandolo come parametro all'opzione, o di ottenere il primo dispositivo libero se non si specifica nessun argomento. Si può poi rimuovere l'associazione e liberare il dispositivo con l'opzione `-d` (che prende come argomento il nome del dispositivo stesso), o ottenere un elenco dei dispositivi definiti con l'opzione `-a`; per l'elenco delle opzioni e tutti i dettagli sul comando si consulti la pagina di manuale.

Ci si può chiedere, a parte l'esempio citato, a cosa possa servire una funzionalità come quella del *loopback*, dato che avere a disposizione un file contenente un filesystem può sembrare un caso abbastanza esoterico. In realtà, dato che in un sistema unix-like tutto è un file, il caso è abbastanza comune; basta infatti copiare con `dd` un file di dispositivo su un file per ottenere l'immagine del contenuto del dispositivo stesso, come illustrato in sez. 4.1.4, ed a questo punto poterne ispezionare il contenuto e verificare l'integrità è senz'altro molto utile.

Come accennato più volte in precedenza nel funzionamento di `mount` è fondamentale il file `/etc/fstab` (il cui nome sta per ***f**ile **s**ystem **tab**le*) che può essere considerato come una specie di file di configurazione del comando. In generale è cura della procedura di installazione creare un file `/etc/fstab` che contenga tutte le informazioni necessarie a montare tutti i filesystem creati sulla base delle scelte sul partizionamento fatte in tale occasione.

Il formato del file è molto semplice: ogni linea definisce un filesystem da montare, ed è composta da sei campi. Linee vuote o che iniziano per "`#`" vengono ignorate, i campi di ogni linea sono separati da spazi o tabulatori. La pagina di manuale, accessibile con `man fstab`, ne spiega i dettagli, un esempio classico del suo contenuto è il seguente:

[6] questo fintanto che non si supera il numero massimo di dispositivi di *loopback* disponibili; il valore di default del limite è 8, ma può essere cambiato fino ad un massimo assoluto di 256 caricando il modulo di kernel che implementa il dispositivo con una opportuna opzione (vedi sez. 5.2.5).

/etc/fstab
```
# /etc/fstab: static file system information.
#
# file system mount point     type    options                       dump  pass
/dev/hdb5     /               ext2    defaults,errors=remount-ro     0       1
/dev/hdb6     none            swap    sw                             0       0
proc          /proc           proc    defaults                       0       0
/dev/fd0      /floppy         auto    defaults,user,noauto           0       0
/dev/cdrom    /cdrom          iso9660 defaults,ro,user,noauto        0       0
/dev/hdb1     /boot           ext2    rw                             0       2
/dev/hda1     /mnt/win        vfat    defaults,user,noauto           0       0
/dev/hdc4     /mnt/zip        auto    defaults,user,noauto           0       0
```

Il primo campo specifica quale è il filesystem da montare che viene normalmente indicato tramite il file di dispositivo su cui lo stesso risiede: nel caso dell'esempio si avevano due dischi rigidi IDE con varie partizioni, indicati con i vari `/dev/hdaN` e `/dev/hdbN`, un floppy, indicato con `/dev/fd0`, uno ZIP su IDE, indicato come `/dev/hdc4`, un CDROM ed un masterizzatore SCSI, indicati rispettivamente con `/dev/cdrom` e `/dev/sr0`, dove nel caso del CDROM si era usato un link simbolico.

Si noti che per il filesystem `/proc`, il cui contenuto non è mantenuto in nessun dispositivo fisico ma viene fornito direttamente dal kernel, si è usata invece la parola chiave `proc`. Se ci fossero stati dei filesystem di rete montati con NFS (vedi sez. 8.4.2) si sarebbero potute avere anche righe del tipo:

/etc/fstab
```
davis.truelite.it:/data /mnt/nfs    nfs    defaults,user,noauto     0       0
```

in cui nel primo campo viene indicata la directory remota da montare.

L'esempio illustrato fa riferimento ad una installazione precedente la versione 2.3 del FHS, che non prevedeva sistemi di rilevazione automatica dei dispositivi rimuovibili e non richiedeva l'uso della directory `/media` per i dispositivi rimuovibili. Oggi l'uso diretto dei file di dispositivo è sempre meno comune ed un esempio più recente di `/etc/fstab` potrebbe essere il seguente estratto, ottenuto da una installazione ordinaria di una Debian Squeeze:

/etc/fstab
```
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
proc            /proc           proc    defaults        0       0
/dev/scd0       /media/cdrom0   udf,iso9660 user,noauto  0       0

# / was on /dev/sda1 during installation
UUID=4edbcf3d-d8bd-49e0-8af5-7f40cea66bb4 /    ext3   errors=remount-ro 0    1
# swap was on /dev/sda5 during installation
UUID=a2331c52-91ab-445b-bc55-7f42a3b9e2e1 none swap   sw              0    0
```

In questo caso si noti come i file di dispositivo siano sostituiti dagli UUID ad essi associati, specificati in forma di assegnazione alla parola chiave `UUID`. Alternativamente si possono indicare i dispositivi con eventuali etichette usando la parola chiave `LABEL` con la stessa sintassi, ottenendo un effetto equivalente all'uso delle opzioni `-U` e `-L` di `mount`, ed anche in questo caso per il funzionamento del sistema deve essere disponibile `/proc/partitions`.

La scelta dell'uso degli UUID al posto dei file di dispositivo è ormai comune in tutte le distribuzioni più recenti che fanno questa scelta per evitare problemi con eventuali variazioni dei nomi dei file di dispositivo dovuta alla loro allocazione dinamica da parte del sistema di *udev*, come può avvenire perché i dischi sono stati rimontati su un diverso canale IDE, o per il passaggio all'uso di *libATA* che fa apparire i dischi IDE come SCSI, o per un diverso ordine di riconoscimento sul bus SCSI che può avvenire con i dischi USB rimuovibili o con l'uso di controller multipli.

Il secondo campo di `/etc/fstab` indica il *mount point*, cioè la directory dove i file contenuti nel dispositivo indicato nel primo campo saranno resi disponibili, e deve essere specificato con il suo pathname assoluto. Se sul dispositivo non esiste un filesystem che deve essere montato, come avviene nel caso di una partizione di *swap* (vedi sez. 5.1.7), al posto del pathname si deve usare la parola chiave `none`.

Il terzo campo indica il tipo di filesystem che sta sul dispositivo che si vuole montare, e deve essere specificato allo stesso modo con cui lo si indica all'opzione `-t` di `mount`, utilizzando gli stessi valori già riportati in tab. 5.2. Si noti però come negli esempi precedenti venga usata anche la parola chiave `swap`, che non compare in tab. 5.2. Essa infatti serve ad indicare che il dispositivo non contiene un filesystem, ma viene usato per la *swap* (vedi sez. 5.1.7).

Il quarto campo di `/etc/fstab` indica le opzioni con cui si deve montare il filesystem, e prende gli stessi valori dell'opzione `-o` di `mount`, riportati in tab. 5.4. Nel caso si usi l'opzione `defaults` la successiva specificazione di un'altra opzione soprassiede il valore di default. Come per `-o` più opzioni vanno specificate di seguito, separate da virgole e senza spazi interposti.

Gli ultimi due campi sono relativi alla manutenzione del filesystem, il quinto campo indica se effettuare il *dump*[7] del filesystem ed in genere viene lasciato a 0 (per attivarlo occorre usare invece 1) mentre il sesto campo indica la sequenza con cui all'avvio viene lanciato il comando `fsck` per controllare lo stato dei dischi (vedi sez. 5.1.5), uno zero indica che il controllo non deve essere eseguito.

L'uso principale di `/etc/fstab` è il controllo del comportamento del comando `mount -a` che viene utilizzato nella procedura di avvio del sistema per montare automaticamente tutte le directory del sistema se queste sono state installate su filesystem separati rispetto alla radice. In questo caso è necessario marcare la riga relativa con l'opzione `auto`. Per i filesystem che non devono essere montati all'avvio (ad esempio i CD-ROM) si deve specificare invece l'opzione `noauto`.

La presenza di una configurazione in `/etc/fstab` permette inoltre di semplificare l'uso di `mount` poiché per i filesystem in esso elencati il comando può essere invocato specificando solo il *mount point* o solo il file di dispositivo. Inoltre questa sintassi consente l'uso di `mount` anche agli utenti normali, i quali potranno montare un dispositivo qualora si siano specificate le opzioni `user`, `group` o `users` nella riga relativa. L'operazione di montaggio di un filesystem è privilegiata e può essere effettuata in modo generico solo dall'amministratore. In questo modo si può permettere agli utenti di montare i propri CD e floppy, senza però consentirgli di modificare il *mount point* o le opzioni di montaggio.

Dal punto di vista dell'amministrazione si potrà avere che fare con `/etc/fstab` tutte le volte che si aggiunge un disco, o un nuovo dispositivo a blocchi, o si cambiano le partizioni. In questo

[7] è un valore utilizzabile solo per i filesystem (attualmente ext2 ed ext3) che supportano il comando di backup `dump` (vedi sez. 4.1.3); se attivato con un valore non nullo verranno salvate le informazioni che consentono a `dump` di eseguire i backup incrementali.

caso occorrerà identificare qual'è il file di dispositivo o l'UUID da usare e scegliere una directory su cui montarlo. Deve poi essere specificato il filesystem da usare (o `auto` se si vuole tentare il riconoscimento automatico) e le eventuali opzioni di montaggio.

Si noti come nel primo esempio per ZIP e floppy si sia consentito agli utenti di montare il filesystem, ma si sia disabilitato il montaggio all'avvio, e pure il controllo dello stato del filesystem, dato che non è detto che il floppy o lo ZIP siano inseriti nel rispettivo lettore. Lo stesso vale per il CDROM e il masterizzatore, per i quali però si è pure aggiunta l'opzione di montaggio in sola lettura.

Le distribuzioni più recenti tendono comunque a gestire i dispositivi rimuovibili attraverso programmi dedicati che sono in grado di rilevare la loro presenza e montarli automaticamente per cui eventuali voci di questo tipo sono assenti nel secondo esempio, in tal caso infatti quello che in genere viene fatto è inserire l'utente in un apposito gruppo che contiene tutti quelli a cui è consentito utilizzare dispositivi rimuovibili (su Debian ed Ubuntu il gruppo è `plugdev`).

Un secondo file collegato all'uso di `mount` è `/etc/mtab`, che contiene l'elenco dei filesystem montati mostrato dal comando quando lo si invoca senza opzioni. Il file viene creato ed aggiornato automaticamente da `mount` nelle sue operazioni e non deve essere modificato manualmente, ma si tenga presente che se la radice è in sola lettura (o se si è usata l'opzione `-n` per eseguire un montaggio) le sue informazioni possono non essere aggiornate.

Le stesse informazioni sono presenti anche all'interno di `/proc/mounts`, ma in questo caso sono mantenute direttamente dal kernel e sono quindi sempre aggiornate, tanto che talvolta si suggerisce di sostituire `/etc/mtab` con un link simbolico a questo file. In realtà in `/etc/mtab` vengono mantenute informazioni aggiuntive, come l'utente che ha montato un filesystem nel caso dell'opzione `user`, che non sarebbe disponibile nel caso di un link simbolico.

Si tenga presente che quando si monta un filesystem su una directory un eventuale contenuto di quest'ultima viene *oscurato* dal contenuto del nuovo filesystem, e non sarà più possibile accedervi fintanto che questo non viene smontato. Se però si sono aperti dei file in essa presenti questi continueranno a funzionare regolarmente in quanto sono visti attraverso il loro *inode* (si ricordi quanto detto in sez. 1.2.2). Inoltre a partire dal kernel 2.4 diventa possibile *impilare* più operazioni di mount sulla stessa directory, che presenterà il contenuto dell'ultimo filesystem montato, valendo quanto appena detto per il contenuto dei precedenti.

In maniera analoga a come lo si è montato, quando non si ha più la necessità di accedere ad un filesystem, questo potrà essere *smontato*. In questo modo diventa possibile rimuovere, nel caso di kernel modulare, (vedi sez. 5.2.5) le eventuali risorse aggiuntive, e liberare la directory utilizzata per il montaggio per un eventuale riutilizzo. Come accennato a partire dalla serie 2.4.x del kernel questo non è più necessario, in quanto si possono *impilare* più montaggi sulla stessa directory, è comunque necessario smontare un device rimovibile come il floppy o il CD, prima di poterlo estrarre e sostituire.

Il comando in questo caso è `umount` (si dice che la `n` si sia persa nei meandri delle prime implementazioni di Unix) che può prendere come argomento sia il *mount point* che il file di dispositivo, e distacca il relativo filesystem dall'albero dei file. Si tenga presente che fintanto che il filesystem è utilizzato questa operazione non viene permessa: questo significa che fintanto che si hanno processi che hanno aperto dei file contenuti nel filesystem, o la cui directory di lavoro (vedi sez. 1.3.2) è nel filesystem, non sarà possibile effettuarne lo smontaggio, e l'uso del comando darà luogo ad un errore di risorsa occupata.

Il problema può risultare particolarmente antipatico quando si ha a che fare con un dispositivo rimovibile che si vuole estrarre, in tal caso però si può ricorrere al comando `lsof` (vedi sez. 2.4.5) per individuare i processi che stanno ancora utilizzando il filesystem e terminarli o cambiargli directory di lavoro (se trattasi di shell). Inoltre dal kernel 2.4.11 è disponibile un *lazy umount*, attivabile con l'opzione `-l`, che distacca immediatamente il filesystem (impedendo ogni ulteriore accesso allo stesso) ma esegue le successive operazioni di pulizia e la chiusura del filesystem soltanto quando tutte le risorse occupate vengono liberate.

Una sintassi alternativa per `umount` è l'uso dell'opzione `-a`, che smonta tutti i filesystem elencati in `/etc/mtab` (tranne, a partire dalla versione 2.7 del comando, `/proc`), in questo caso ovviamente non è necessario indicare quale dispositivo smontare, ma si può restringere le operazioni a tutti i filesystem di un determinato tipo, specificando quest'ultimo con l'opzione `-t`. L'elenco completo delle opzioni del comando è disponibile nella relativa pagina di manuale, accessibile con `man umount`.

Si tenga presente che quando si smonta un filesystem tutte le modifiche effettuate fino a quel momento dovranno essere registrate su disco. Questo significa, dato che il kernel prevede la bufferizzazione dei dati,[8] che in certe occasioni l'uso di `umount` può risultare piuttosto lento e scatenare una forte attività di accesso al disco, dato che il comando non termina fintanto che tutti i buffer sono stati scaricati. Questa operazione può essere forzata, senza dover smontare il filesystem, con il comando `sync` che si limita semplicemente a richiedere al kernel che lo scarico dei buffer sia effettuato immediatamente.

Altri comandi essenziali per la gestione dei dischi sono quelli che consentono di verificare lo spazio disco utilizzato e la percentuale di uso dei vari filesystem. Il primo di questi è `df`; se invocato senza argomenti il comando stampa l'occupazione dello spazio disco di ciascun filesystem montato, con un sommario della relativa dimensione totale, dei blocchi occupati e liberi e della percentuale di occupazione.

Il comando si cura di riportare soltanto quelli che hanno un contenuto reale, non vengono cioè mostrati filesystem virtuali come `/proc` o `/sys` che non hanno una reale occupazione di spazio disco o di altre risorse. Un esempio del possibile risultato è qualcosa del tipo:

```
piccardi@hain:~$ df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/md0             114389928  82038556  26540668  76% /
tmpfs                   258436         0    258436   0% /lib/init/rw
udev                     10240        64     10176   1% /dev
tmpfs                   258436         4    258432   1% /dev/shm
/dev/sda1            192292124 115097092  67427112  64% /bacula
```

Il default prevede che il comando indichi l'occupazione di spazio disco in blocchi da 1K (1024 byte), ma si possono richiedere dimensioni diverse o la stampa in un formato più leggibile con l'opzione `-h` che invece di riportare i numeri per esteso usa alla bisogna i suffissi dei multipli decimali (K, M, G) e scrive le dimensioni in byte rendendo più evidenti i numeri in gioco.

Si può inoltre indicare al comando un singolo filesystem, per ottenere i dati relativi solo a quello o un tipo tramite l'opzione `-t`, che prende come parametro uno del valori di tab. 5.2. Infine

[8] dato che l'accesso a disco è lento il kernel cerca di eseguire i trasferimenti dei dati nella maniera più efficace possibile, salvando le modifiche in memoria ed effettuando le scritture in gruppo al momento reputato più opportuno.

con l'opzione `-i` si può richiedere un rapporto relativo all'uso degli *inode* invece che a quello dello spazio disco. Al solito per i dettagli e tutte le opzioni si consulti la pagina di manuale.

Qualora invece del totale a livello di intero filesystem si voglia conoscere lo spazio disco occupato dai file posti all'interno di una certa directory si può usare il comando `du`. Il comando prende come argomenti una lista di file e directory e stampa un sommario dello spazio occupato da ciascuno di essi, in kilobyte. Se non si specificano argomenti viene usata la directory corrente.

Il comando stampa ricorsivamente la quantità di spazio disco occupato all'interno di ciascuna sottodirectory a partire da quelle indicate come argomenti, se si vuole conoscere solo il totale relativo alle directory indicate si può utilizzare l'opzione `-s`. Una seconda opzione utile è `-c` che permette, quando si indicano più argomenti, di ottenere anche la stampa dell'occupazione totale relativa a tutti gli argomenti. Il comando, come `df`, supporta anche l'opzione `-h` per la stampa dei numeri con le abbreviazioni. Per i dettagli sul funzionamento del comando (cambiare unità di misura, escludere file dal calcolo, ecc.) si consulti al solito la pagina di manuale.

5.1.4 La creazione di un filesystem

Abbiamo visto in sez. 5.1.3 come si può rendere disponibile il contenuto di un filesystem nell'albero delle directory. Quando si installa il sistema o quando si aggiunge un disco però i filesystem devono essere creati, in modo da strutturare per l'uso lo spazio disponibile (che sia una partizione o un dispositivo fisico o virtuale), operazione comunemente detta *formattazione*.

La creazione di un filesystem su un dispositivo si esegue con il comando `mkfs`, questo prende come unico argomento il file di dispositivo su cui si vuole creare il filesystem, mentre il tipo di filesystem che si vuole creare si specifica con l'opzione `-t`. Fa eccezione il caso di *btrfs* che consente di specificare più dispositivi (come vedremo più avanti questo filesystem ha caratteristiche più evolute rispetto a quelli tradizionali).

In realtà `mkfs` è solo un comando generico, che chiama per ciascun tipo di filesystem, indicato con l'opzione `-t` nella forma `-t type` dove `type` è uno dei valori di tab.5.2, uno specifico sottocomando `mkfs.type`. Così ad esempio se si vuole formattare un filesystem FAT per Windows, usando `mkfs -t vfat`, verrà invocato il comando `mkfs.vfat`.

In generale i programmi di formattazione sono forniti insieme agli altri programmi di supporto per il filesystem, come quelli per il controllo di consistenza o il ridimensionamento, e possono avere un loro nome specifico: ad esempio per *ReiserFS* il programma si chiama `mkreiserfs` mentre quello per *ext2/ext3/ext4* è `mke2fs`, e possono anche essere invocati direttamente. Perché la forma generica `mkfs -t type` funzioni però deve essere presente il corrispondente comando `mkfs.type`, cosa che nel caso di filesystem come XFS e *btrfs* è comunque automaticamente vero, dato che i relativi comandi nativi di formattazione sono proprio `mkfs.xfs` e `mkfs.btrfs`.

Tutte le altre opzioni da passare ad `mkfs` dipendono dal filesystem scelto, e possono essere trovate nella pagina di manuale del relativo comando specifico, accessibile al solito con `man mkfs.type`. Un elenco delle principali opzioni disponibili per i principali filesystem usati su Linux è riportato in tab. 5.5.

Nella "*formattazione*" di un nuovo disco la prima scelta da fare è quella del tipo di filesystem che si vuole usare, cosa che dipende principalmente dall'utilizzo che se ne farà. Ad esempio se si vuole formattare un una chiavetta USB, la scelta più comune è l'uso di un filesystem VFAT per garantire la leggibilità dello stesso anche da Windows, visto che l'uso più comune di queste chiavette è proprio quello per lo scambio di dati.

Tipo	Opzione	Descrizione
reiserfs		mkreiserfs
	-b *N*	specifica la dimensione dei blocchi.
	-j *file*	specifica un file di dispositivo per il giornale.
	-l *label*	imposta una etichetta sul filesystem (16 caratteri).
	-s *N*	specifica la dimensione del giornale.
	-u *UUID*	associa un UUID al filesystem.
msdos		mkdosfs o mkfs.msdos
	-F *N*	specifica la dimensione della FAT (*File Allocation Table*).
	-n *name*	specifica un nome per il volume (11 caratteri).
vfat		mkfs.vfat
		esegue mkdosfs.
ext2		mke2fs o mkfs.ext2
	-b *N*	specifica la dimensione dei blocchi.
	-F	forza successivo controllo del filesystem.
	-i *N*	specifica ogni quanti byte di spazio disco deve essere creato un *inode*, non può essere inferiore alle dimensioni di un blocco.
	-j	crea il giornale per il filesystem (equivalente ad invocare il comando come mkfs.ext3).
	-J *opts*	permette di specificare i parametri per la gestione del file di giornale.
	-L *label*	associa un'etichetta al filesystem.
	-m *N*	specifica la percentuale di spazio disco riservata per l'amministratore che gli utenti normali non possono usare.
	-T *type*	specifica la tipologia di utilizzo del filesystem, con un'indicazione generica che consente di stabilire lo spazio da assegnare ad *inode* e spazio per i file.
	-U *UUID*	associa un UUID al filesystem.
ext3		mkfs.ext3
		esegue mke2fs con le opportune opzioni.
ext4		mkfs.ext4
		esegue mke2fs con le opportune opzioni.
xfs		mkfs.xfs
	-b *N*	specifica la dimensione dei blocchi.
	-d *opt*	imposta le opzioni per la gestione dei dati dei file.
	-f	forza la sovrascrittura di un filesystem già presente.
	-i *opt*	imposta le opzioni per la gestione degli *inode*.
	-l *opt*	imposta le opzioni per il *journaling* del filesystem.
	-L *label*	associa un'etichetta al filesystem (12 caratteri).
	-m *opt*	imposta le opzioni di gestione generali per i metadati del filesystem.
	-n *opt*	imposta le opzioni di gestione per le directory.
	-N	stampa i parametri del filesystem che verrebbe creato, senza scrivere sul disco.
	-s *N*	specifica la dimensione dei settori.
btrfs		mkfs.btrfs
	-b *N*	specifica la dimensione del filesystem (in byte).
	-d *type*	specifica la distribuzione dei dati (prende come valori uno fra: raid0, raid1, raid5, raid6, raid10 o single).
	-L *label*	associa un'etichetta al filesystem.
	-U *UUID*	associa un UUID al filesystem.

Tabella 5.5: Principali comandi, e relative opzioni, per la creazione dei filesystem.

Se però il disco deve contenere dei dati del sistema gestiti con Linux questa scelta non è adatta, non disponendo quel filesystem del supporto necessario per utenti e permessi. In tal caso le considerazioni che di devono fare attengono principalmente alle prestazioni ed alle funzionalità, ma soprattutto alla stabilità ed affidabilità del filesystem, che è il requisito fondamentale dato

che scopo principale di un filesystem è quello di mantenere i dati. Per questo motivo ogni distribuzione sceglie quello che ritiene essere il "migliore" come filesystem di default.

Benché oggi esistano molteplici alternative con diverse caratteristiche, il filesystem più usato su Linux è *ext4*, evoluzione del primo filesystem nativo, di cui comunque resta ancora piuttosto utilizzata anche la versione precedente *ext3* ed in misura minore anche la originaria *ext2*. Tutte queste versioni usano una strutturazione dei dati su disco analoga, per cui alla fine i vari programmi di manutenzione sono gli stessi e mantengono i nomi originari nati con *ext2*. Per questo motivo, e per la maggior diffusione, li tratteremo in maniera più estesa rispetto agli altri filesystem. Il primo di questi comandi è quello di creazione, `mke2fs`, un cui esempio è:

```
anarres:/home/piccardi# mke2fs /dev/sdd1
mke2fs 1.42.2 (9-Apr-2012)
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
Stride=0 blocks, Stripe width=0 blocks
65024 inodes, 259680 blocks
12984 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=67371008
32 block groups
8192 blocks per group, 8192 fragments per group
2032 inodes per group
Superblock backups stored on blocks:
        8193, 24577, 40961, 57345, 73729, 204801, 221185

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
```

Se usato senza specificare altro che il dispositivo da formattare, `mke2fs` si limita a creare il filesystem, stampando tutta una serie di informazioni ad esso relative. È particolarmente significativa quella relativa a dove sono stati posti i vari backup del *superblock*.[9] Questo è un blocco speciale che contiene tutte le informazioni relative alla configurazione del filesystem, ed è essenziale per poterlo montare. La perdita del *superblock* causerebbe la perdita completa del filesystem per cui ne vengono mantenute varie copie in modo che in caso di distruzione o corruzione si possa usare una delle copie. Pertanto è bene annotarsi la posizione delle varie copie, anche se è possibile recuperarla in un secondo momento con `dumpe2fs`.

Benché normalmente il comando richieda come argomento un file di dispositivo, volendo si può anche usare un file normale per crearvi un filesystem che potrà poi essere montato in *loopback*. In tal caso però il comando, accorgendosi di non avere a che fare con un disco (anzi per essere precisi con un dispositivo a blocchi), chiede esplicitamente se proseguire o meno.

Al di là della invocazione semplice appena mostrata il comando prevede numerose opzioni che permettono di impostare tutta una serie di caratteristiche del filesystem che si va a creare. Una delle principali è la dimensione dei *blocchi* in cui lo spazio disco viene suddiviso, da specificare

[9] la spiegazione dettagliata del significato dei vari parametri, come quelli relativi a *group blocks* e *fragments*, va al di là dello scopo di questo testo, ma in breve si può dire che un filesystem *ext2* divide lo spazio disco in blocchi raggruppati nei *group blocks*, che contengono al loro interno le tabelle degli *inode* e lo spazio per i blocchi di dati e le informazioni per evitare la frammentazione dello spazio disco; per una spiegazione più completa si può leggere il file `filesystem/ext2.txt` distribuito nella documentazione allegata ai sorgenti del kernel.

tramite l'opzione -b. Con *blocco* si indica in genere l'unità minima di spazio disco che viene allocata per inserirvi i dati di un file; in genere, anche se per alcuni filesystem più evoluti esistono dei meccanismi automatici che permettono di mettere il contenuto di diversi file di piccole dimensioni in un solo blocco, ogni file consuma almeno un blocco, e le dimensioni dei file sono dei multipli interi di questo valore. Valori tipici sono 1024, 2048 o 4096.

Qualora si ometta questo parametro il suo valore viene stabilito con una valutazione euristica sulla base della dimensione del disco e del tipo di uso che si vuole fare del filesystem. Questo può essere indicato tramite l'opzione -T, che prende i tre valori news (un *inode* ogni 4kb), largefile (un *inode* ogni 1Mb) e largefile4 (un *inode* ogni 4Mb) per assegnare un diverso rapporto fra numero di *inode* e spazio disco disponibile. Qualora si usi un valore negativo questo viene preso come limite inferiore per la dimensione di un blocco.

Un altro parametro importante è quello del numero di *inode* da creare, si ricordi infatti che, come illustrato in sez. 1.2.2, ogni file è sempre associato ad un *inode*. Questi di norma vengono mantenuti in una sezione apposita del filesystem, le cui dimensioni, nel caso di *ext2*/*ext3*/*ext4* vengono determinate in sede di creazione e non possono più essere modificate in un secondo tempo a meno di non espandere le dimensioni del filesystem.

Il comando permette, con l'uso dell'opzione -i di impostare ogni quanti byte di spazio disco deve essere creato un *inode*. Si deve dare cioè la valutazione della dimensione media di un file sul disco: specificare un valore troppo alto creerà pochi *inode*, con il rischio di esaurirli, un valore troppo basso ne creerà troppi, con conseguente spreco di risorse e soprattutto un maggior lavoro per le operazioni sugli stessi, che dovranno essere eseguite su intervalli più ampi.

Si tenga presente che per la gestione dei file in un filesystem blocchi e *inode* sono due risorse indipendenti, si possono cioè esaurire sia i blocchi liberi che gli *inode* disponibili, anche se il primo caso è di gran lunga quello più comune, e trovarsi così nella condizione di non poter più creare nuovi file. Per questo motivo per una gestione ottimale del disco di solito occorre una scelta opportuna sia delle dimensioni dei blocchi che del rapporto fra blocchi disponibili e *inode*. Tenere le dimensioni dei blocchi basse riduce lo spreco di spazio per quelli occupati parzialmente, ma implica un maggiore lavoro per la gestione (e maggiore spazio per le informazioni relative). Usare pochi *inode* permette di risparmiare spazio ed essere più veloci nella ricerca, ma si corre il rischio di finirli prima di esaurire lo spazio disco.

Dato che la struttura di base è sempre la stessa, quello che contraddistingue le varie versioni della famiglia di filesystem *extN* sono le funzionalità che vengono abilitate. Queste possono essere impostate singolarmente in fase di creazione attraverso l'opzione -O, che prende una lista di parametri, separata da virgole se sono più di uno, che specificano le funzionalità da abilitare. Se si fa precedere un parametro da un accento circonflesso "^" si richiede invece la disabilitazione della relativa funzionalità, inoltre l'uso del valore speciale none permette di disabilitare tutte le caratteristiche aggiuntive.

I valori possibili per i parametri di -O sono illustrati in tab. 5.6, ed in realtà quello che contraddistingue le differenze fra le varie versioni è proprio quali di queste funzionalità vengono abilitate, le più avanzate delle quali sono disponibili solo su *ext4*. Nelle vecchie versioni del comando di default venivano attivate solo le due caratteristiche sparse_super e filetype, che comunque non sono compatibili con le implementazioni di *ext2* antecedenti i kernel della serie 2.2.x. Oggi il default è controllato dalla voce base_features del file /etc/mke2fs.conf, per la cui sintassi si rimanda alla pagina di manuale. Queste in genere sono solo le funzionalità più elementari per cui se si usa semplicemente mke2fs il filesystem viene formattato come *ext2*.

Parametro	Significato
dir_index	usa una struttura ad alberi ed hash per la gestione del contenuto delle directory.
extent	usa una diversa e più efficiente modalità di allocazione dei dati, disponibile solo per *ext4*.
filetype	memorizza il tipo di file nelle voci delle directory.
has_journal	crea anche il file per il giornale (equivalente all'uso di -j).
journal_dev	richiede che sul dispositivo indicato sia creato un giornale invece di un filesystem.
large_files	il filesystem può contenere file di dimensione superiore a 2Gb (attivato automaticamente nei kernel recenti alla creazione di un tale file).
resize_inode	riserva dello spazio per consentire un successivo allargamento del filesystem con resize2fs (vedi sez. 6.3.2).
sparse_super	crea un filesystem con meno copie del *superblock* per non sprecare spazio sui filesystem molto grandi.

Tabella 5.6: Principali parametri per l'attivazione delle funzionalità dei filesystem *ext2*/*ext3*/*ext4* attivabili mediante l'opzione -O di mke2fs.

Nel caso si voglia creare un filesystem di tipo *ext3*, si deve utilizzare l'opzione -j o invocare il comando come mkfs.ext3, in tal caso vengono attivate anche le ulteriori funzionalità riportate nella configurazione /etc/mke2fs.conf che prevede la possibilità di indicare queste (e quelle relative ad *ext4*) all'interno di altrettante direttive di configurazione specifiche. In questo caso la funzionalità aggiuntiva in questione è has_journal, che permette di creare un file apposito per il giornale (affronteremo il tema dei filesystem che supportano il *journaling* in sez. 5.1.5).

Quando con mke2fs si attiva il giornale, indipendentemente dalle modalità utilizzate, si può poi utilizzare l'ulteriore opzione -J che permette di specificarne le caratteristiche. Se questa non viene utilizzata sono utilizzati i valori di default, altrimenti questi possono essere specificati nella forma parametro=valore. Un elenco dei parametri specificabili con -J è illustrato in tab. 5.7, con relativa spiegazione.

Parametro	Significato
device	permette di impostare un dispositivo esterno per mantenere il giornale. Al posto di un file di dispositivo si può specificare una l'etichetta associata al filesystem (con l'opzione -L). Il dispositivo che fa da giornale deve essere inizializzato con mke2fs -O journal_dev.
size	specifica la dimensione del giornale, da specificare in megabyte. Deve essere un minimo di 1024 blocchi, e non può superare 102400 blocchi.

Tabella 5.7: Parametri per la gestione del giornale di un filesystem *ext3* specificabili tramite l'opzione -J di mke2fs.

Qualora si voglia creare un filesystem *ext4*, che ha ulteriori funzionalità avanzate, come l'uso dei cosiddetti *extent* che consente una allocazione più efficiente e veloce dei dati, ed opzioni di gestione dei metadati che consentono controllo e successiva riparazione del filesystem in caso di problemi molto più veloce (vedi sez. 5.1.5) occorre invece usare il comando come mkfs.ext4.

Una volta creato il filesystem si possono modificarne alcune caratteristiche in un secondo tempo con l'uso del comando tune2fs. L'uso più comune di tune2fs è probabilmente quello per trasformare un filesystem *ext2* in un filesystem *ext3* creando il giornale. Questo è ottenibile

immediatamente con l'opzione `-j`, e si può usare `-J` con le stesse opzioni di tab. 5.7 per indicare le caratteristiche del giornale.

Opzione	**Significato**
`-C` *N*	modifica il contatore del numero di volte che un filesystem è stato montato.
`-c` *N*	imposta il numero di volte che un filesystem può essere rimontato prima di subire un controllo.
`-e` *val*	modifica il comportamento del kernel quando viene rilevato un errore sul filesystem, prende come parametro uno dei valori: `continue` (continua ignorando l'errore), `remount-ro` (rimonta il filesystem in sola lettura), `panic` (si ferma causando un *kernel panic*).
`-g` *grp*	imposta il gruppo che può usare i blocchi riservati.
`-i` *time*	imposta l'intervallo di tempo fra due controlli successivi; prende come parametro un numero di giorni, di settimane se si pospone il carattere "`w`", di mesi se si pospone il carattere "`m`".
`-j`	aggiunge un giornale per il filesystem.
`-J` *val*	sovrascrive i parametri del giornale, prende come parametro uno dei valori di tab. 5.7.
`-L` *labl*	imposta il nome del volume, per un massimo di 16 caratteri.
`-l`	stampa i contenuti del *superblock* (come se si eseguisse `dumpe2fs`), vedi sez. 5.1.5).
`-m` *N*	imposta la percentuale di blocchi riservati.
`-O` *val*	imposta le caratteristiche avanzate del filesystem, prende come parametri uno dei valori di tab. 5.6.
`-r` *N*	imposta il numero di blocchi riservati.
`-T` *time*	modifica il valore del tempo in cui il filesystem è stato verificato per l'ultima volta; prende come parametro l'indicazione della data nel formato usato da `date` o `now` per indicare il tempo corrente.
`-U` *uuid*	imposta l'UUID del filesystem.
`-u` *user*	imposta l'utente che può usare i blocchi riservati.

Tabella 5.8: Principali opzioni per il comando `tune2fs`.

Un'altra opzione di uso comune è `-c` che permette di impostare il numero di volte che il filesystem può essere montato senza che venga forzato un controllo di consistenza con `fsck`, analoga a questa è `-i` che specifica l'intervallo massimo, in giorni, mesi o settimane, fra due controlli. Con `-C` si può anche modificare a mano il contatore del numero di volte che un filesystem è stato montato, in modo da poter forzare il controllo ad un successivo riavvio.

Con l'opzione `-O` si possono modificare in un secondo tempo le caratteristiche avanzate specificando come parametro una lista dei valori già illustrati in tab. 5.6, in questo caso per disabilitare una funzionalità si può apporre un carattere "`^`" al corrispondente parametro. Si tenga presente che se si modificano o abilitano alcune funzionalità (questo avviene se ad esempio si attivano quelle per *ext4*) per poterle davvero utilizzare quasi sempre è necessario eseguire preventivamente una riparazione/ricostruzione del filesystem con `e2fsck`.

Infine si può usare l'opzione `-L` per impostare una etichetta sul filesystem, che viene a costituire il nome del volume. Analogamente si può modificare il valore dell'UUID associato al filesystem con l'opzione `-U`, cui deve essere passato come parametro il nuovo valore. Diventa così possibile, come visto in sez. 5.1.3, fare riferimento al filesystem in `/etc/fstab` usando l'etichetta

o l'UUID al posto del file di dispositivo. Le altre opzioni principali sono illustrate in tab. 5.8, al solito si può ottenere l'elenco completo ed una descrizione dettagliata dalla pagina di manuale del comando.

Per quanto riguarda le possibili alternative a *ext4*, ad oggi *ReiserFS*, che è stato il primo a disporre del *journaling* e che per questo ottenne una buona adozione, è ormai in sostanziale disuso: lo sviluppo infatti è fermo alla versione 3 per la sola correzione degli errori, la successiva versione 4 infatti è stata rifiutata dagli sviluppatori del kernel e praticamente non viene più sviluppata.

Anche lo sviluppo di *ext4* è sostanzialmente terminato in termine di nuove funzionalità, in quanto il nuovo filesystem nativo di Linux dovrebbe essere *btrfs*, che presenta una serie di funzionalità avanzate, come la possibilità di essere suddiviso su un numero arbitrario di dispositivi (rendendo superfluo LVM), il supporto nativo a replicazioni di tipo RAID (vedi sez. 6.1.1), la possibilità di creare *snapshot* e sotto-volumi da montare come filesystem separati, e molto altro. Il problema però è che al momento non è ancora considerato sufficientemente stabile e nessuna distribuzione lo ha adottato come default.

Per questo motivo la sola alternativa a *ext4* che fornisca tutte le garanzie di stabilità e affidabilità necessarie è *XFS*, che presenta anche altri vantaggi, ed il cui sviluppo prosegue sia in termini di miglioramento di prestazioni che di aggiunta di nuove funzionalità. La sua diffusione sta crescendo anche in ragione del fatto e che è stato adottato come default da varie distribuzioni (in particolare da RedHat con versione 7 della sua "*Enterprise Linux*").

A parte quelli in termini di prestazioni, che pur nominalmente migliori (specie per file di grandi dimensioni) sono spesso ambigui perché dipendono dal tipo di utilizzo del filesystem,[10] il principale vantaggio di XFS rispetto ad *ext4* è che non esiste più la rigida suddivisione fra spazio per gli *inode* e spazio per i blocchi per cui una allocazione iniziale può essere modificata in un tempo successivo. Inoltre il filesystem è in grado di auto-ripararsi a caldo e non prevede, tranne casi gravi ed evidenti di corruzione, la necessità di un controllo periodico all'avvio. Insieme ai vantaggi il filesystem presenta anche qualche svantaggio, come il fatto che può essere ridimensionato solo in accrescimento.

Come accennato il comando di creazione di un filesystem XFS è `mkfs.xfs`. Senza opzioni, a meno che non rilevi la presenza di un altro filesystem, nel qual caso è necessario applicare l'opzione `-f`, il comando si limita alla creazione del filesystem sul dispositivo indicato come argomento, applicando una serie di default per le varie caratteristiche dello stesso, che poi stampa a video con qualcosa del tipo:

```
root@monk:~# mkfs.xfs /dev/sdb1
meta-data=/dev/sdb1              isize=256    agcount=4, agsize=15262348 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=0        finobt=0
data     =                       bsize=4096   blocks=61049389, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=0
log      =internal log           bsize=4096   blocks=29809, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

[10]un punto debole del filesystem era la velocità di scrittura per i metadati degli *inode*, ma a partire dal kernel 3.2 il problema è stato risolto e le prestazioni secondo gli sviluppatori sono anche superiori a quelle dei concorrenti.

Il comando riporta i valori dei parametri usati per le varie caratteristiche di gestione del filesystem, elencando in successione quelli per i metadati generali usati dal filesystem (sezione *meta-data*), quelli per i dati dei file (sezione *data*), quelli per i contenuti e l'indicizzazione delle directory (sezione *naming*), quelli per il sistema del *journaling* (sezione *log*) e quelli della gestione dell'I/O in *realtime*[11] (sezione *realtime*).

Queste caratteristiche possono essere modificate in fase di creazione del filesystem usando rispettivamente le opzioni -m, -d, -n, -l e -r del comando, cui si aggiunge -i per impostare le caratteristiche degli *inode*. Tutte queste opzioni richiedono che si passi un argomento con l'indicazione di parametri e relativi valori come lista di assegnazioni nella forma *chiave=valore* separate da virgole; alternativamente si può ripetere più volte l'uso di una opzione per impostare parametri diversi.

Chiave	**Significato**
Caratteristiche per gli *inode* (opzione -i)	
maxpct	indica la percentuale massima di spazio del filesystem che può essere allocata per gli *inode*, il default è il 25% per i filesystem sotto un terabyte e decresce al 5% per quelli fino a 50 terabyte e all'1% sopra.
Caratteristiche per i dati (opzione -d)	
file	indica se il filesystem deve essere creato su un file ordinario (valore 1) o no (valore 0), se non specificato si assume 0, se specificato senza assegnazione si intende 1 e questo è necessario per creare l'immagine di un filesystem XFS su un file.
name	nel caso si sia richiesta la creazione del filesystem su un file ordinario, indica il nome dello stesso; quando si è usata questa opzione il giornale deve essere necessariamente interno.
size	indica le dimensioni della sezione dati del filesystem, la si usa quando si vuol creare un'immagine su un file ordinario, per indicarne le dimensioni, o se si vuole lasciare dello spazio libero in coda ad un dispositivo.
Caratteristiche del *journaling* (opzione -l)	
logdev	imposta il dispositivo su cui mantenere il giornale del filesystem, richiede che si indichi il relativo file di dispositivo.
size	imposta la dimensione del giornale, richiede un valore numerico.
internal	prende i valori 0 e 1, e quest'ultimo, il default, indica che il giornale viene gestito insieme ai dati; se specificato esplicitamente è incompatibile con l'uso di logdev.

Tabella 5.9: Principali caratteristiche del filesystem impostabili con mkfs.xfs.

In genere se non ci sono necessità specifiche, non si ha ragione per modificare i default, ma se ad esempio si vuole indicare una specifica dimensione per il giornale, occorrerà usare l'opzione -l ed usare la parola chiave size per dichiararla. I valori dei parametri dipendono ovviamente dalla caratteristica che si vuole impostare, ma sono per lo più numerici e quando indicano delle dimensioni relative a spazio disco supportano i suffissi s, b, k, m, g, t (e via a salire) per indicare rispettivamente numero di settori, byte, kilobyte, megabyte, gigabyte, terabyte, ecc.

[11] la versione originaria di IRIX, in presenza di hardware specializzato, consentiva di fornire un rate di I/O garantito ai programmi *realtime* che ne avevano bisogno, questa garanzia non è disponibile su Linux, che comunque non è un sistema operativo *realtime*.

Si sono riportati quelli più rilevanti in tab.5.9, per i dettagli si rimanda alla pagina di manuale accessibile con `man mkfs.xfs`.

Opzione	Significato
`-f`	da usare quando il filesystem è contenuto in un file e non in un dispositivo.
`-l`	stampa l'etichetta associata al filesystem.
`-L` *labl*	imposta l'etichetta associata al filesystem.
`-u`	stampa l'UUID associato al filesystem.
`-U` *uuid*	imposta l'UUID associato al filesystem.

Tabella 5.10: Principali opzioni di `xfs_admin`.

Una volta creato il filesystem si possono modificare alcune delle sue proprietà con il comando `xfs_admin`, che è l'analogo di `tune2fs`, ma a differenza di quest'ultimo richiede che il filesystem sia smontato se deve scriverci. Il programma, che richiede come argomento il dispositivo su cui è mantenuto il filesystem, consente sia di leggere che di impostare l'etichetta e l'UUID del filesystem.[12] Buona parte delle altre caratteristiche del filesystem possono essere modificate a caldo con `xfs_growfs`, che tratteremo in sez. 6.3.2. In tab. 5.10 si sono riportate le principali opzioni di `xfs_admin`, per i dettagli e l'elenco completo si rimanda alla pagina di manuale.

5.1.5 Controllo e riparazione di un filesystem

In generale il funzionamento dei filesystem in Linux è estremamente stabile, ed una volta creati è praticamente impossibile danneggiarli nel corso delle normali operazioni, a meno di non usare versioni sperimentali del kernel (ma in questo caso si stanno cercando rogne, ed è possibile trovarle). Se però si ha un black-out improvviso, qualcuno inciampa nel cavo di alimentazione del vostro PC, o a donna delle pulizia spegne il quadro elettrico, è normale che il filesystem, dal momento in cui l'aggiornamento dei dati su disco è stato interrotto brutalmente, si possa trovare in uno stato incoerente.

In questo caso si può avere un danneggiamento della struttura del filesystem, che deve essere riparato. In genere ogni filesystem prevede l'esistenza di un flag su disco, attivato quando viene montato, che indica che è in uso, e che viene azzerato solo quando il filesystem viene smontato, nel qual caso si è certi che tutte le operazioni sospese sono state completate e che lo stato è coerente. Se c'è stata un'interruzione della corrente, o un qualunque altro problema che ha comportato un riavvio non pulito, questo flag resterà attivo ed il sistema potrà rilevare, al successivo tentativo di montaggio, che qualcosa è andato storto.

Quello che può succedere in questi casi dipende dal tipo di filesystem. Coi filesystem tradizionali `mount` rileva l'errore e non monta il filesystem o lo monta in sola lettura (a seconda delle opzioni scelte). A questo punto occorre usare l'opportuno programma di controllo per verificare lo stato del filesystem ed eventualmente riparare gli errori. Di norma, in caso di rilevamento di un errore durante la procedura di avvio del sistema, questo viene lanciato automaticamente secondo quanto specificato in `/etc/fstab` (vedi sez. 5.1.3).

[12] a differenza di *ext4*, XFS considera l'UUID del filesystem una proprietà intrinseca dello stesso, e si rifiuta di montare un filesystem se un altro con lo stesso UUID è già montato, per questo motivo non è possibile modificarlo a filesystem montato.

Il programma generico per il controllo di un filesystem è `fsck` (da ***file system check***) che come `mkfs` non è altro che un front-end per i singoli programmi specifici di ciascun tipo di filesystem. Questi vengono attivati attraverso l'opzione `-t` seguita dal tipo del filesystem (quello di tab.5.2). Il programma prende come argomenti un elenco di filesystem da controllare specificati per dispositivo, o anche per *mount point*, se sono citati in `/etc/fstab`. Se si indica più di un filesystem la lista dei relativi tipi, separata da virgole, deve essere specificata come parametro per `-t`.

Quando viene invocato con l'opzione `-A` (di solito questo viene fatto nella procedura di avvio) il comando esegue una scansione di `/etc/fstab` e cerca di controllare ogni filesystem ivi elencato (utilizzando il terzo campo per identificarne il tipo), a meno che il sesto campo del file non sia impostato a zero (si ricordi quanto detto in sez. 5.1.3). Il comando prima proverà ad eseguire il controllo per il filesystem della directory radice e passerà poi a tutti gli altri in ordine di valore crescente del suddetto campo.

Come per `mkfs` le opzioni disponibili per `fsck` dipendono dallo specifico filesystem, a parte quelle generiche relative alla gestione della sequenza delle scansioni (per cui si rimanda alla pagina di manuale) il comando passa tutte le opzioni che non riconosce al programma specifico del filesystem selezionato. Una convenzione vorrebbe che i programmi di controllo definiscano alcune opzioni generiche, indicando con `-a` di eseguire le riparazioni in modo automatico, senza chiedere l'intervento dell'amministratore, e con `-r` di eseguire le riparazioni in modalità interattiva, ma non tutti i programmi che vedremo lo fanno.

Di norma l'esecuzione automatica del controllo prevede che il filesystem sia riparato senza necessità di intervento da parte dell'utente. Ci sono casi comunque in cui questo non è possibile, nel qual caso il programma si ferma per chiedere lumi all'utente. Quando questo avviene durante la procedura di avvio di norma il sistema viene portato in *single user mode* (si veda sez. 5.3.5) e viene richiesto di rieseguire il programma manualmente. In tal caso di norma il filesystem non è montato, a meno che il filesystem danneggiato non sia la radice, nel qual caso esso deve essere comunque montato, ma in sola lettura. Chiaramente se il filesystem radice è danneggiato così gravemente da non poter neanche essere montato in sola lettura, si avrà un *kernel panic*.

La procedura di controllo e riparazione può essere molto lunga e complessa, specie per filesystem di grandi dimensioni, in quanto prevede una serie di verifiche dettagliate per identificare informazioni incoerenti e parziali, che comportano varie scansioni dell'intero contenuto del filesystem stesso. Questo può significare dei tempi che possono diventare ore o addirittura giorni per i dischi più grandi. Per ovviare a questo inconveniente alcuni filesystem più avanzati supportano il cosiddetto *journaling*, un sistema che permette di riportare il filesystem in uno stato coerente con grande velocità.

Il concetto fondamentale del *journaling* è che le operazioni sul filesystem vengono prima registrate su un *giornale*, un file speciale a questo dedicato, in genere invisibile e mantenuto direttamente dal kernel, e poi riportate sul filesystem. Così se si ha una interruzione improvvisa si hanno due casi: se l'interruzione è avvenuta durante l'aggiornamento del filesystem, il giornale sarà intatto e lo si potrà utilizzare per completare l'aggiornamento interrotto. Se invece l'interruzione è avvenuta durante la scrittura nel giornale sarà questo ad essere scartato, si perderanno così le ultime modifiche, ma il filesystem resterà in uno stato coerente.

In questo modo quando si ha un *filesystem journaled* si può evitare il lungo procedimento di riparazione. In realtà non è che il procedimento non avvenga, solo che grazie alla presenza del giornale questo viene eseguito con grande rapidità, non dovendo effettuare il controllo completo

del filesystem. In generale però, anche se si usa il *journaling*, è opportuno mantenere un controllo periodico sul filesystem, in quanto errori sul disco, cavi difettosi o bug del kernel potrebbero comunque aver corrotto le informazioni.

Con Linux tutti i filesystem più recenti supportano nativamente il *journaling*, a parte il vecchio *ext2* che ha ottenuto questa funzionalità con la successiva versione *ext3*. L'uso di un giornale ha però un impatto significativo sulle prestazioni del filesystem, e non è detto che sia sempre utilizzato, tanto che con il successivo *ext4* è stata prevista la possibilità di disabilitarlo, cosa che continua ad avere senso per i filesystem che possono essere utilizzati in sola lettura e per quelli per cui il recupero veloce all'avvio non è essenziale (ad esempio un disco su cui si effettuano copie di backup).

Nel caso dei filesystem tradizionali della famiglia *extN* l'uso del giornale di realizza in maniera abbastanza semplice, per *ext2* la conversione ad *ext3* è immediata, basta aggiungere il giornale al filesystem con `tune2fs -j` e rimontarlo come *ext3*. Il passaggio da *ext3* ad *ext4* è ancora più semplice, basta rimontare *ext3* come *ext4*, anche se questo abiliterà solo le funzioni compatibili all'indietro e molte delle funzionalità più avanzate di *ext4* non saranno disponibili.

Una conversione più completa si può ottenere invece abilitando le funzionalità avanzate che servono con `tune2fs` (ad esempio forzando le opzioni `extent,uninit_bg,dir_index` con `-O`) ed effettuando appunto un controllo forzato del filesystem, che provvederà ad effettuare tutte le modifiche e correzioni necessarie, che in questo caso non dipendono da una corruzione ma dall'aggiornamento. Questo però comporta che il filesystem sia smontato cosa che rende le cose più problematiche per la radice, che non può essere smontata, ma è comunque fattibile operando sul filesystem rimontato in sola lettura.

Dato che si può sostanzialmente considerare la famiglia dei vari *ext2*, *ext3* e *ext4* come il filesystem più diffuso su Linux, tratteremo in maniera più estesa i programmi di gestione specifici di questa famiglia, che sono comuni a tutte le varie versioni. Il programma di controllo e riparazione del filesystem è `e2fsck`, che prende come argomento il dispositivo da controllare.

Il comando supporta le opzioni generiche `-a` e `-r` illustrate in precedenza, che sono state mantenute solo per compatibilità, in realtà `-a` è deprecata in favore della più recente `-p` mentre `-r` non fa niente dato che il comando viene sempre eseguito in modalità interattiva. È comunque possibile usare l'opzione `-y` per far rispondere "*yes*" automaticamente a tutte le domande in modo da poter eseguire il comando in modalità non interattiva (ad esempio da uno script), mentre con `-n` si fa la stessa cosa aprendo il filesystem in sola lettura.

In caso di filesystem pesantemente corrotto, o di cui si è cancellata la parte iniziale, cosa che può capitare se ad esempio si è sbagliato il dispositivo di uscita di un `dd`, si possono poi specificare il *superblock* da utilizzare con `-b` e la dimensione di un blocco con `-B`. Se si è usato un giornale posto su un altro dispositivo questo deve essere specificato con l'opzione `-j`. Quando si è usata l'opzione `-b` e il filesystem non è stato aperto in sola lettura `e2fsck` si cura anche di ripristinare il superblock al completamento del controllo. Le altre opzioni principali del comando sono riportate in tab. 5.11, per un elenco completo e la relativa documentazione si può al solito fare riferimento alla pagina di manuale del comando.

In genere non c'è necessità di eseguire direttamente `e2fsck`, in quanto di norma questo viene eseguito dagli script di avvio. Nel caso lo si voglia eseguire comunque occorre assicurarsi che il filesystem relativo non sia montato, o sia montato in sola lettura, ed inoltre specificare l'opzione `-f` senza la quale il comando non controlla un filesystem che per lui risulta pulito.

Opzione	Significato
`-b` *N*	specifica un superblocco alternativo.
`-B` *size*	specifica le dimensioni di un blocco.
`-c`	esegue il controllo del dispositivo per eventuali settori difettosi.
`-f`	forza il controllo del filesystem.
`-j` *file*	specifica il dispositivo di un giornale esterno.
`-l` *file*	specifica il file con una lista di blocchi difettosi da aggiungere a quelli già presenti.
`-L` *file*	specifica il file con la nuova lista dei blocchi difettosi (sovrascrivendo quella presente).
`-n`	esegue il controllo rispondendo automaticamente "*no*" a tutte le domande.
`-t`	stampa delle statistiche di esecuzione.
`-y`	esegue il controllo rispondendo automaticamente "*yes*" a tutte le domande.

Tabella 5.11: Principali opzioni per il comando `e2fsck`.

Nel caso si usi *ReiserFS* il programma che consente di eseguire controllo e riparazione di un filesystem (quello invocato con `fsck.reiserfs`) è `reiserfsck`. Il programma deve essere usato su un filesystem smontato, anche se alcune opzioni, come `--check`, possono essere usate su un filesystem montato in sola lettura. Il comando richiede come argomento obbligatorio il file di dispositivo contenente il filesystem da controllare. Le opzioni principali sono riportate in tab. 5.12, per i dettagli si consulti al solito la pagina di manuale.

Opzione	Significato
`--check`	esegue un controllo di consistenza e riporta i risultati, senza eseguire nessuna riparazione.
`--fix-fixable`	esegue le riparazioni relative a problemi minori effettuabili senza dover effettuare una ricostruzione completa dell'intero filesystem.
`--rebuild-tree`	ricostruisce l'intero filesystem dai dati presenti sul disco, da usare solo in caso di danni gravi riportati da `--check` e previo backup della partizione contenente il filesystem in quanto potenzialmente distruttiva.
`-a, -p`	opzioni usate nel controllo automatico all'avvio, causano un controllo dello stato del filesystem e la successiva esecuzione con l'opzione `--fix-fixable` se vengono riportati danni minori.
`-B`	indica il file contenente la lista dei blocchi difettosi.
`-f`	forza l'esecuzione del comando anche se il filesystem risulta a posto.

Tabella 5.12: Principali opzioni di `reiserfsck`.

Nel caso di XFS il comportamento dei programmi di riparazione e controllo, a causa delle diverse modalità di funzionamento di questo filesystem, è piuttosto diverso rispetto ai precedenti. Ad esempio il comando `fsck.xfs`, pur esistendo per compatibilità con l'uso di `fsck`, non fa assolutamente niente ed ha sempre successo. Il controllo di consistenza del filesystem infatti non viene effettuato all'avvio, ma direttamente quando il filesystem viene montato, ed eventuali riparazioni relative a corruzioni minori vengono eseguite automaticamente in quell'occasione senza dover utilizzare un comando specifico.

Qualora si voglia effettuare una verifica manuale è comunque possibile eseguire direttamente il comando `xfs_repair`, che prende come argomento il dispositivo contenente il filesystem. Il

comando richiede che questo sia smontato o al più montato in sola lettura; in quest'ultimo caso, necessario se si tratta della radice, è necessario indicare l'opzione `-d` ed è richiesto un immediato riavvio dopo la terminazione del comando. A meno di eseguirlo in modalità prolissa con l'opzione `-v`, il programma non scrive nulla se non in caso di problemi. Le altre opzioni principali sono elencate in tab. 5.13.

Opzione	**Significato**
`-d`	esegue la riparazione in modalità definita pericolosa a filesystem montato in sola lettura, necessaria quando il filesystem è quello della radice.
`-f`	da usare quando il filesystem è contenuto in un file e non in un dispositivo.
`-l` *dev*	indica il dispositivo del giornale se questo è esterno.
`-n`	esegue la scansione senza eseguire nessuna modifica.
`-v`	modalità prolissa, produce una quantità di informazioni eccessiva ed è intesa ad uso di debug.

Tabella 5.13: Principali opzioni di `xfs_repair`.

Benché il sistema sia molto stabile ed i principali problemi di corruzione possano essere risolti con i vari comandi di controllo appena trattati, in caso di danni veramente seri (le cui cause normalmente sono l'uso di un kernel sperimentale o problemi hardware) questi possono fallire. Esistono pertanto degli ulteriori comandi diagnostici che possono risultare utili per cercare di risolvere la situazione; di nuovo tratteremo con maggior dettaglio quelli disponibili per la famiglia dei filesystem *extN*, anche se ne esistono di analoghi anche per tutti gli altri.

Come già accennato in sez. 5.1.4 il comando `dumpe2fs` può essere usato per recuperare una serie di informazioni riguardo ad un filesystem *extN*. Se invocato senza opzioni il comando stampa una lunga lista di informazioni ricavate dal contenuto del *superblock* e dei *group block*, come esempio riportiamo solo la prima parte dell'output del comando, quella relativa alle sole informazioni del *superblock*, ottenibili specificando l'opzione `-h`:

```
anarres:~# dumpe2fs -h /dev/sdd4
dumpe2fs 1.35-WIP (07-Dec-2003)
Filesystem volume name:   <none>
Last mounted on:          <not available>
Filesystem UUID:          d0cb4773-dbf8-4898-94e8-bb2acc41df0d
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal filetype needs_recovery sparse_super
Default mount options:    (none)
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              4169760
Block count:              8324194
Reserved block count:     416209
Free blocks:              5365536
Free inodes:              3935324
First block:              0
Block size:               4096
Fragment size:            4096
Blocks per group:         32768
Fragments per group:      32768
Inodes per group:         16352
```

```
Inode blocks per group:   511
Last mount time:          Fri Jan 30 20:59:45 2004
Last write time:          Fri Jan 30 20:59:45 2004
Mount count:              16
Maximum mount count:      22
Last checked:             Sun Dec 21 14:45:22 2003
Check interval:           15552000 (6 months)
Next check after:         Fri Jun 18 15:45:22 2004
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:               128
Journal inode:            13
First orphan inode:       2421098
Journal backup:           inode blocks
```

Le altre opzioni principali del comando sono `-b`, che restituisce l'elenco dei blocchi marchiati come danneggiati in un filesystem, `-ob` che permette di specificare il blocco (indicato per numero) da usare come *superblock* al posto di quello standard (in caso di corruzione di quest'ultimo), `-oB` per indicare la dimensione dei blocchi (anche questa è una informazione necessaria solo in caso di corruzione del filesystem). Infine `-i` permette di leggere le informazioni invece che dal filesystem da una immagine creata con il comando `e2image`. Per un elenco completo e relative spiegazioni si faccia al solito riferimento alla pagina di manuale.

Una delle misure di precauzione che si possono prendere per tentare un recupero in caso di corruzione di un filesystem *ext2* è quella di crearne una immagine con il comando `e2image`. Questo comando permette di salvare su un file i dati critici del filesystem in modo da poterli riutilizzare con programmi di riparazione come `e2fsck` o `debugfs`. Il comando prende come argomenti il dispositivo su cui si trova il filesystem ed il nome del file su cui salvare l'immagine. L'unica opzione è `-r` che crea una immagine binaria che può essere usata dai vari programmi di controllo come se fosse l'intero filesystem.[13]

Qualora `e2fsck` fallisca in caso di un filesystem pesantemente danneggiato un'ultima risorsa che si può tentare è quella di utilizzare `debugfs` per provare ad eseguire manualmente la riparazione. Il comando permette di aprire anche un filesystem danneggiato, e di eseguire su di esso delle operazioni. Non è detto che si riesca a riparare completamente il filesystem, dato che per questo occorre una conoscenza dettagliata della sua struttura interna, ma si possono tentare delle operazioni di ripulitura che potrebbero portare lo stesso in uno stato gestibile da `e2fsck`. In effetti il programma, come dice il suo nome, non è uno strumento per il controllo quanto per il debugging, ed il suo uso è in genere riservato ai cosiddetti *filesystem guru*.

Il comando prende come argomento il dispositivo contenente il filesystem da esaminare; questo viene sempre aperto in sola lettura a meno che non si specifichi l'opzione `-w`. In caso di filesystem pesantemente danneggiato l'opzione `-b` permette di specificare una dimensione dei blocchi, l'opzione `-s` permette di specificare il *superblock*. Le opzioni principali sono riportate in tab. 5.14, per l'elenco completo al solito si consulti la pagina di manuale.

Una volta aperto un filesystem con `debugfs` si viene portati su una riga di comando da cui diventa possibile operare a basso livello con i dati del filesystem, compiendo anche operazioni

[13]per far questo viene creato uno *sparse file* delle stesse dimensioni del filesystem; uno *sparse file* è un file in cui non sono state scritte le parti vuote, pertanto anche se la sua dimensione può essere enorme, pari appunto ad un intero filesystem, in realtà viene occupato su disco solo lo spazio relativo alle parti non vuote.

Opzione	**Significato**
`-b` *`N`*	specifica la dimensione dei blocchi.
`-c`	apre il filesystem in modalità *catastrofe*, in cui non vengono letti neanche i dati generali dello stesso.
`-f` *`file`*	legge i comandi dal file passato come parametro.
`-s` *`N`*	specifica un superblocco alternativo.
`-w`	apre il filesystem in lettura/scrittura.

Tabella 5.14: Principali opzioni per il comando `debugfs`.

impossibili con i normali comandi dei file, come cancellare o creare un *hard link* senza aggiornare il numero di riferimenti ad un *inode*, recuperare un file cancellato ecc. Molti dei comandi del programma riprendono i nomi degli ordinari comandi di shell, eseguendo compiti analoghi, ma operano direttamente sui dati del filesystem e possono usare come argomenti sia dei normali nomi di file che dei numeri di *inode*, questi ultimi però devono essere indicati esplicitamente fra parentesi angolari (<>). Si è riportato un breve riassunto dei comandi principali in tab. 5.15, per i dettagli si consulti come sempre la pagina di manuale.

Comando	**Significato**
`cat`	mostra il contenuto di un file.
`cd`	cambia directory di lavoro.
`dump`	salva i dati del file/*inode* su un altro file.
`ln`	crea un *hard link* ma non aggiorna il numero di collegamenti.
`ls`	mostra la lista dei file (con i dati degli *inode*).
`lsdel`	mostra la lista dei file cancellati.
`rm`	cancella un file.
`rmdir`	cancella una directory.
`stat`	stampa i dati di un *inode*.
`undel`	ripristina un file cancellato.
`unlink`	rimuove il collegamento ad un *inode* ma non aggiorna il numero di collegamenti.

Tabella 5.15: Principali comandi di `debugfs`.

Anche per XFS esistono ulteriori programmi diagnostici, ad esempio come analogo di `dumpe2fs` si può usare il comando `xfs_info` che stampa le informazioni relative ad un filesystem e richiede che si indichi il file di dispositivo.[14] Maggiori informazioni relative ai metadati del filesystem possono essere ottenuto con `xfs_metadump`, che scrive su un file tutti i metadati, ma il programma viene utilizzato principalmente a scopo di debug, per fornire agli sviluppatori i dati di un filesystem non riparabile con `xfs_repair` offuscando opportunamente i contenuti (nomi dei file, directory ecc.). Infine come analogo di `debugfs` è disponibile `xfs_db`, anch'esso è principalmente uno strumento di analisi di basso livello, che richiede una approfondita conoscenza delle strutture del filesystem, da usare (leggendo la documentazione) come ultima risorsa qualora `xfs_repair` fallisse.

Infine si tenga presente che per Linux in genere non si usano programmi per la *deframmentazione* del disco. Questa infatti è un problema solo per quei filesystem la cui architettura è

[14] in realtà questo è un alias per una opzione del comando di ridimensionamento (equivalente a `xfs_growfs -n`), che vedremo in sez. 6.3.2.

talmente inadeguata da renderlo tale. Normalmente un filesystem ben progettato è in grado di gestire la allocazione dello spazio disco in maniera da evitare il sorgere del problema fin dall'inizio. Ci sono comunque alcuni programmi per eseguire questa operazione o altre ottimizzazioni; ad esempio per XFS c'è `xfs_fsr`, che agisce a filesystem attivo ed esegue le operazioni in maniera parallela all'uso corrente dello stesso, ma sia questo che gli altri non vengono usati quasi mai, proprio perché di scarsa utilità.

5.1.6 Il sistema dell'*automounter*

Abbiamo visto in sez. 5.1.3 come sia possibile impostare grazie a `/etc/fstab` il montaggio all'avvio di tutti i filesystem necessari, e in quella occasione abbiamo anche accennato anche come la gran parte dei sistemi desktop sia in grado di eseguire automaticamente il montaggio di eventuali dispositivi estraibili quando il loro uso viene rilevato dal kernel, funzionalità che oggi viene gestita da appositi programmi che si appoggiano ad *udev* (vedi sez. 5.4.5) e che non ha nulla a che fare con quanto prenderemo in esame qui.

Esiste però un'altra esigenza relativa al montaggio automatico dei filesystem, che non attiene a dispositivi estraibili o temporanei, ma piuttosto a filesystem sempre disponibili, ma utilizzati solo sporadicamente, come potrebbero essere filesystem di rete per accedere a informazioni condivise. In questo caso può essere utile poter avere un montaggio "a richiesta", soltanto in occasione del loro utilizzo. É a questa esigenza che risponde il sistema dell'*automounter* il cui scopo è proprio quello di montare automaticamente un filesystem in reazione all'accesso alla directory in cui questo deve comparire.

Il sistema consente in questo modo di rimandare al momento del loro uso effettivo il montaggio di filesystem di rete senza la necessità di dover eseguire un comando in maniera esplicita. Questo consente anche di evitare intoppi nella procedura di avvio, quando in caso di malfunzionamento della rete detti filesystem non potrebbero venire montati. Infine essendo stata prevista la possibilità di mantenere le informazioni relative ai filesystem da montare automaticamente su servizi di rete centralizzati (sono supportati NIS e LDAP, ma trattare il loro uso va al di là degli obiettivi di questo testo) si può semplificare la gestione in caso si debbano configurare allo stesso modo molte macchine.

Il supporto per l'*automounter*, denominato *autofs*, viene fornito direttamente da parte dal kernel a partire dalla versione 2.2. Il relativo supporto deve comunque essere stato compilato all'interno del kernel (vedi sez. 5.2.3) ed è inoltre necessario l'ausilio di alcuni programmi in user-space.[15] In realtà esiste anche un supporto totalmente in user-space, fornito da un apposito demone (`amd`, o *auto mount daemon*) ripreso da quello usato su Solaris, che però non prenderemo in considerazione.

L'intero sistema è controllato da un file di configurazione principale, `/etc/auto.master`, che contiene l'elenco dei *mount-point* su cui verranno montati automaticamente i filesystem. Il formato del file, con la solita convenzione di ignorare righe vuote e tutto quello che segue il carattere "`#`" prevede che ogni riga indichi un *mount-point* a cui viene associata una *mappa* che descrive quali filesystem vi devono essere montati e delle eventuali opzioni; il tutto deve essere specificato da tre campi separati da spazi.

Fino alla versione 4 di *autofs* erano supportate soltanto le cosiddette *mappe indirette*, in cui il campo del *mount-point* deve indicare una directory sul filesystem (ad esempio `/misc`) sotto cui

[15] esistono due versioni, la 4 e la 5, su Debian vengono fornite rispettivamente dai pacchetti `autofs` e `autofs5`.

compariranno tutti i filesystem elencati nella mappa corrispondente. In sostanza con una mappa indiretta l'accesso ad una sottodirectory (ad esempio sotto `/misc/disk`) provoca una ricerca nella mappa associata con la chiave espressa dal nome della sottodirectory (nel caso dell'esempio `disk`) che se trovata provoca l'esecuzione della rispettiva operazione di montaggio.

Con la versione 5 di *autofs* è divenuto disponibile anche il supporto per le cosiddette *mappe dirette*, in questo caso il `mount-point` viene indicato come "`/-`" (che non corrisponde alla directory "`-`" sotto la radice) mentre le chiavi usate nella mappa diretta devono corrispondere ai pathname assoluti dove si vuole che i rispettivi filesystem vengano montati. Si tenga presente che in entrambi i casi se le directory indicate non esistono queste verranno create automaticamente al montaggio del filesystem che devono ospitare, e rimosse al suo smontaggio.

Il secondo campo di `/etc/auto.master` serve ad indicare la mappa associata al rispettivo *mount-point*. Nella sua forma più semplice questa è espressa semplicemente come il pathname del file che la contiene. In realtà come accennato le mappe possono essere mantenute anche tramite servizi di rete, per cui la sintassi estesa di questo secondo campo è nella forma `tipo:mappa`,[16] dove `tipo` indica di che mappa si tratta e `nome` deve essere espresso in forma adeguata al relativo tipo, ma non tratteremo queste forme più complesse, per le quali si rimanda alla lettura della pagina di manuale con accessibile `man auto.master`.

Il terzo campo di `auto.master` consente di specificare delle opzioni facoltative. Qualora queste siano indicate senza un trattino iniziale verranno semplicemente passate come opzioni per l'esecuzione del comando `mount` per i filesystem della mappa, il trattino iniziale consente invece di passare opzioni relative al funzionamento della mappa. Anche in questo caso per i dettagli si consulti la pagina di manuale.

Si tenga presente inoltre che esiste, a partire dalla versione 5, una speciale mappa interna, che non richiede riferimenti ad altre risorse, utilizzabile specificando `-host` nel relativo campo. Questa mappa consente di montare automaticamente un qualunque filesystem NFS disponibile sulla propria rete; in tal caso la sottodirectory del *mount-point* verrà interpretata come un nome di una macchina sulla rete, ed al di sotto di detta sottodirectory verranno montati tutti i filesystem esportati con NFS da quella macchina. In questo modo non è più necessario mantenere una mappa con l'elenco degli stessi.

Un possibile esempio del contenuto di `/etc/auto.master` è il seguente, dove si è richiesto il montaggio automatico sotto `/misc` dei filesystem elencati in `/etc/auto.misc` e in `/net` dei filesystem NFS disponibili nella propria rete:

/etc/auto.master

```
/misc  /etc/auto.misc
/net   -host
```

L'avvio del sistema dell'*automounter* è effettuato dallo script `/etc/init.d/autofs` che esamina il contenuto di `/etc/auto.master` e per ciascuna voce esegue il programma `automount`, come singole istanze del programma nella versione 4, come *thread* aggiuntivo di una unica istanza nella versione 5. Questo è il programma che si incarica, all'accesso alle directory indicate in `/etc/auto.master`, di eseguire la scansione delle relative mappe e di effettuare il montaggio automatico. Ad ogni modifica di `/etc/auto.master` occorre eseguire un reload del servizio per far

[16] ad essere pignoli la sintassi completa è `tipo,formato:mappa`, ma in tutti i casi pratici il default del formato è sempre quello corrispondente al tipo di mappa, quindi non viene mai indicato.

rieseguire automount sulle voci presenti, le istanze relative a voci rimosse verranno eliminate e ne verranno fatte partire di nuove per le voci aggiunte.

Invece una modifica di un file delle mappe (ad /etc/auto.misc nel caso dell'esempio) viene vista immediatamente al primo utilizzo successivo dello stesso. Inoltre è possibile modificare alcune delle impostazioni generali del sistema che normalmente vengono mantenute nel file /etc/default/autofs, in cui queste sono espresse in forma di assegnazione di variabili di shell. Si sono riportate in tab. 5.16 le più rilevanti, l'elenco completo si trova al solito nella pagina di manuale di auto.master.

Variabile	**Significato**
TIMEOUT	tempo in secondi dopo il quale un filesystem inutilizzato viene smontato automaticamente.
APPEND_OPTIONS	determina se le opzioni di montaggio inserite nelle mappe vengono aggiunte (default yes) o se rimpiazzano quelle eventualmente specificate in auto.master.
LOGGING	indica il livello di log.

Tabella 5.16: Variabili di configurazione per i default di *autofs*.

Il formato di un file di una mappa *autofs* prevede di tre campi separati da spazi, di cui il secondo è opzionale. Il primo campo è la stringa che costituisce la chiave di ricerca sulla mappa, e che come detto corrisponde alla sottodirectory della voce di auto.master corrispondente alla mappa su cui verrà montato il filesystem indicato nell'ultimo campo.

Il secondo campo, che se presente deve iniziare con il carattere "-", indica le opzioni, espresse da una lista separata da virgole, da usare nel montaggio del filesystem che devono corrispondere ai parametri dell'opzione -o che si userebbero per montare il filesystem con il comando mount, con l'eccezione dell'opzione speciale -fstype=tipo che serve invece ad esprimere il tipo di filesystem, con tipo che deve corrispondere al valore che si darebbe all'opzione -t del comando mount. Se non si specifica -fstype il default è usare nfs, e se non si devono specificare opzioni il secondo campo si può omettere.

L'ultimo campo indica invece il filesystem da montare, nel caso di NFS questo deve essere espresso nella forma hostname:pathname nello stesso formato che si userebbe per il primo campo di /etc/fstab. Qualora si voglia invece montare il contenuto di un dispositivo locale lo si dovrà indicare tramite il corrispondente file di dispositivo preceduto dal carattere ":", qualora di tratti di una condivisione di rete con protocollo SMB invece la si dovrà indicare, sempre preceduta dal carattere ":", con la convenzione per gli indirizzi usata dal comando smbclient (per la quale si rimanda a sez. 8.4.3). Sono inoltre supportate alcune funzionalità avanzate, come l'uso di montaggi multipli, e la sostituzione di alcune variabili per generare mappe dinamiche, per la trattazione delle quali si rimanda alla pagina di manuale del formato delle mappe, accessibile con man 5 autofs.

5.1.7 La gestione della *swap* e dei CDROM

Tratteremo in questa ultima sezione due argomenti attinenti solo in modo parziale alla gestione di dischi e filesystem. Il primo di questi è quello relativo alla gestione dello spazio disco dedicato alla *swap* ed al funzionamento di quest'ultima. Questa costituisce un complemento alla gestione della memoria virtuale, il meccanismo con cui il kernel controlla l'allocazione della memoria fisica ai vari processi.

Come accennato in sez. 1.1 e poi approfondito in sez. 1.3.2 il kernel gestisce, con l'aiuto della MMU (*Memory Management Unit*) del processore, una rimappatura dello spazio degli indirizzi dei processi sulla memoria fisica effettivamente disponibile. In questo modo ciascun processo può usare un suo spazio di indirizzi virtuale (usando un qualunque valore fra quelli possibili), e tramite la MMU accederà ad un indirizzo reale della memoria effettivamente disponibile.

Una delle caratteristiche del meccanismo è che in questo modo ciascun processo mantiene un suo spazio di indirizzi separato e se un processo usa un indirizzo che non corrisponde a nessuna pagina di memoria si avrà quello che si chiama un *segmentation fault* ed il sistema si accorgerà immediatamente dell'errore inviando al processo un segnale di `SIGSEGV`. Ma la potenza del meccanismo della memoria virtuale consiste nel fatto che esso consente di utilizzare anche dello spazio disco come supporto aggiuntivo rispetto alla memoria fisica, quando l'uso di questa diventa eccessivo.

È questo lo scopo della cosiddetta *area di swap*, una sezione di disco (in genere una partizione, ma si può usare anche un file) su cui il kernel può salvare le pagine di memoria meno utilizzate di un processo, ma anche i dati mantenuti nei dischi in RAM generati con `tmpfs`, in modo che altri possano utilizzare la memoria fisica che così viene liberata. Chiaramente quando si andrà a rieseguire il processo le cui pagine di memoria sono state salvate sull'*area di swap* e questo cercherà di accedere a quelle, si avrà quello che si chiama un *page fault*[17] a questo punto il sistema della memoria virtuale provvederà a recuperare la pagina dall'area di *swap* e a rimetterla in memoria, così che il processo possa proseguire la sua esecuzione come se niente fosse. In realtà tutto questo richiede tempi che sono ordini di grandezza superiori rispetto all'accesso diretto alla RAM, per cui il niente è solo teorico ed in pratica il programma sarà infinitamente più lento.

Perché il kernel possa usare un'*area di swap* questa deve essere opportunamente inizializzata, come si fa per un filesystem, ma si tenga presente comunque che in questo caso non si ha a che fare con un filesystem, dato che l'area deve essere solo utilizzata per copiarci pagine, per cui non sono affatto necessarie tutte le infrastrutture mostrate in sez. 1.2.2. A questo provvede il comando `mkswap` che prende come argomento il file da usare come area di *swap*. In genere si usa il comando specificando come argomento un file di dispositivo indicante una partizione appositamente riservata (vedi sez. 5.1.2), in questo caso infatti l'accesso al disco è diretto e più veloce rispetto al caso, comunque possibile, in cui si usa un file normale, per il quale si dovrebbe prima passare attraverso il relativo filesystem.

Dato che le prestazioni del sistema della memoria virtuale dipendono direttamente, in caso di utilizzo dell'area di *swap*, dai tempi di accesso a quest'ultima e dalla velocità di trasferimento dei dati, è in genere una pessima idea usare un disco vecchio e lento per questo scopo. È per questo motivo inoltre che a volte si suggerisce di usare per l'area di *swap* l'ultima partizione disponibile, considerata come quella che contiene i settori più esterni del disco, dove la velocità di trasferimento è maggiore.

Il comando prevede, per compatibilità con le vecchie versioni, un secondo argomento che specifica la dimensione dell'area di *swap* e se non lo si specifica viene automaticamente utilizzato tutto lo spazio disponibile. Come per `mkfs` l'opzione `-c` richiede un controllo della partizione per la presenza di settori difettosi. L'opzione `-p` permette di specificare come parametro la di-

[17] quello che succede è che quando la MMU si accorge che la pagina richiesta non è mappata su della memoria fisica invia un segnale al processore, così che questo possa eseguire la opportuna sezione del kernel, quest'ultimo si occuperà o di recuperare la pagina dalla *swap* (se l'indirizzo era giusto) o di inviare un segnale di `SIGSEGV` (se l'accesso era sbagliato).

mensione delle pagine di memoria (in genere 4096 byte, ma dipende dall'architettura hardware), l'elenco delle opzioni è riportato in tab. 5.17.

Opzione	Significato
`-c`	esegue un controllo per la presenza di settori difettosi.
`-f`	forza l'esecuzione del comando anche se si sono dati argomenti sbagliati (come un dimensione dell'area maggiore di quella della partizione).
`-p` *`size`*	specifica, con il valore passato come parametro, la dimensione delle pagine di memoria.
`-v` *`N`*	specifica la versione della formattazione dell'area di *swap*, con `-v0` usa il vecchio stile (deprecato), con il `-v1` usa il nuovo.

Tabella 5.17: Opzioni per il comando `mkswap`.

Si tenga presente che per i kernel precedenti il 2.4.10 erano possibili fino ad un massimo di 8 aree di *swap*, a partire da esso sono state portate a 32. Un elenco di quelle attive è visibile in `/proc/swaps`. La dimensione massima dell'area di *swap* dipende sia dalla versione utilizzata che dall'architettura hardware. Con la vecchia versione il massimo dipendeva solo dalla dimensione delle pagine ed era di 128Mb con le pagine di 4096 byte della architettura standard dei PC portati poi a 2Gb, oggi è di 64Gb.

Una volta creata un'area di *swap* questa non verrà utilizzata dal kernel fintanto che non la si attiva con il comando `swapon`; questo prende come argomento il file da usare coma area di *swap*, che verrà immediatamente attivata. In generale però il comando non viene mai invocato direttamente, ma chiamato con l'opzione `-a` negli script di avvio, ed in questo caso attiverà tutti i dispositivi che sono marcati con `swap` all'interno di `/etc/fstab`.[18] Se invece si vuole avere una lista dei dispositivi attualmente utilizzati si potrà usare l'opzione `-s`, ottenendo qualcosa del tipo:

```
hain:/home/piccardi/truedoc/corso# swapon -s
Filename                        Type            Size    Used    Priority
/dev/hda2                       partition       498004  1936    -1
```

Un'altra opzione importante è `-p`, che permette di impostare una priorità (un parametro di valore fra 0 e 32767) per l'uso della partizione. Quando si ha una sola area questo valore non è significativo in quanto il kernel si limiterà ad usare la prima sezione libera che trova, se le aree sono più di una l'algoritmo di utilizzo prevede che il kernel usi a turno quelle della stessa priorità (realizzando così una forma di RAID-0, vedi sez. 6.1.1) e passi ad una di priorità inferiore solo quando quelle di priorità superiore sono piene.

Se non si specifica nulla le priorità vengono assegnate automaticamente andando a diminuire nell'ordine di attivazione (così che la prima area attivata è quella a priorità maggiore), usando `-p` si può forzare la stessa priorità ottenendo un uso più efficace e maggiore velocità. Si può ottenere questo risultato anche per le aree di *swap* attivate con `swapon -a` specificando in `/etc/fstab` nel relativo campo l'opzione `pri=`*`N`* dove *`N`*, è lo stesso valore che si userebbe con `-p`.

Infine si può disattivare un'area di *swap* usando il comando `swapoff`, anche in questo caso occorre fornire come argomento il file di dispositivo indicante l'area da disabilitare. Il comando

[18] si ricordi di nuovo che la *swap* non è un filesystem, pertanto anche se la voce ad essa relativa compare in `/etc/fstab`, sintassi, come già visto in sez. 5.1.3, è diversa, e che non si può usare il comando `mount` per montarne il contenuto in una directory.

consente anche l'uso dell'opzione `-a` nel qual caso non sarà necessario passare argomenti e saranno disattivate tutte le aree di *swap* presenti, così come indicate in `/proc/swaps` e `/etc/fstab`.

Benché totalmente diversi dai dischi, anche i CDROM sono dispositivi a blocchi, e come visto in sez. 5.1.3 per poter accedere al loro contenuto occorre che questi vengano opportunamente montati. Abbiamo anche accennato come in genere detto contenuto sia mantenuto in uno specifico filesystem di tipo `iso9660`, definito nello standard omonimo. Si tenga presente comunque che in questo caso stiamo parlando dei CD di dati, i CD audio non contengono un filesystem e si utilizzano in maniera totalmente diversa, che non approfondiremo qui.

Il problema che però non abbiamo affrontato in sez. 5.1.4 è come creare un tale filesystem e come registrarlo su un CDROM quando si vuole crearne uno. In questo caso infatti, trattandosi di un dispositivo fisico diverso, non esiste il concetto delle partizioni, e se l'accesso diretto in lettura può essere fatto, come per dischi, tramite il file di dispositivo, non altrettanto accade per una eventuale scrittura con un masterizzatore. Questo vuol dire che anche se è sempre possibile leggere l'immagine di un disco con un comando come `dd if=/dev/cdrom of=file.iso` non è possibile creare un CDROM con qualcosa del tipo `dd if=file.iso of=/dev/cdrom`, anche se si ha un masterizzatore e non un semplice lettore.

Il comando che consente la creazione di un filesystem `iso9660` è `mkisofs`, anche se questo è il nome del programma originale, sempre disponibile come link, ma la gran parte delle distribuzioni oggi usa la versione sviluppata dopo un fork, che si chiama `genisoimage`. Per quanto appena detto a differenza dei vari `mkfs` questo non può essere usato direttamente con un file di dispositivo,[19] ma serve a creare una *immagine* dello stesso su un file, che è quello che potrà essere usato in seguito con un programma di masterizzazione.

La sintassi del comando prevede che si passino come argomenti le directory che si vogliono includere nell'immagine, in genere si usa sempre l'opzione `-o` per specificare il file dell'immagine ISO, altrimenti il comando scrive sullo *standard output*. Il comando prevede inoltre una sterminata quantità di opzioni che consentono di controllare le modalità di creazione dell'immagine e supportare le innumerevoli varianti delle estensioni fatte allo standard originale, che essendo nato come minimo comun denominatore delle caratteristiche dei filesystem dell'epoca, supporta funzionalità minime, derivate per lo più dalle limitazioni dovuto al dover essere compatibile con l'MSDOS del tempo, come ad esempio nomi dei file di al massimo otto caratteri più tre di estensione, oltre ovviamente a non supportare nessuna forma di permessi o proprietà dei file, link simbolici, ecc.

Dato che con filesystem più recenti questo costituirebbe un limite scarsamente accettabile, sono state introdotte diverse estensioni, come le *Rock Ridge*, attivabili con l'opzione `-R`, che consentono di crearvi un vero filesystem Unix (con permessi, proprietari dei file, link simbolici, nomi estesi, ecc.) o le *Joliet*, attivabili con l'opzione `-J`, ed usate nei sistemi Windows per estendere i nomi dei file. Le due opzioni possono essere combinate per aver il supporto di entrambe le funzionalità. Si possono inoltre creare immagini adatte a sistemi Macintosh con le estensioni HFS, abilitate con l'opzione `-hfs`.

Oltre ai semplici CD di dati il comando è anche in grado di creare CDROM avviabili in diversi formati, e ad esempio si può usare l'opzione `-b` per indicare come parametro il file contenente una immagine di disco di avvio per PC secondo le estensioni *El Torito*, che in sostanza non è altro che l'immagine di un floppy di avvio, creata ad esempio con SYSLINUX o ISOLINUX (vedi

[19] a meno di non voler formattare una partizione o un disco con questo filesystem, cosa che non ha alcuna utilità pratica.

Opzione	Significato
`-b` *image*	crea una immagine con incluso il disco di avvio `image` secondo le estensioni *El Torito*.
`-hfs`	crea una immagine con le estensioni *HFS* per l'uso da parte di sistemi Macintosh.
`-J`	crea una immagine con le estensioni *Joliet* per l'uso da parte di sistemi Windows.
`-m` *glob*	esclude dai file inseriti nell'immagine quelli selezionati con l'espressione `glob` (indicata con la sintassi del *filename globbing* della shell).
`-o` *file*	indica il file su cui scrivere l'immagine ISO.
`-R`	crea una immagine con le estensioni *Rock Ridge* per l'uso da parte di sistemi unix-like.

Tabella 5.18: Opzioni per il comando `mkisofs`.

sez. 5.3.2), che può avere dimensioni fisse di 1200, 1440, 2880 kb. Le altre opzioni più rilevanti si sono riportate in tab. 5.18, ed al solito per i dettagli e delle spiegazioni più approfondite si consulti la pagina di manuale.

5.2 La gestione di kernel e moduli

Tratteremo in questa sezione la gestione del kernel, in tutti i suoi aspetti: dalla scelta delle diverse versioni, al tipo di kernel da utilizzare, la sua ricompilazione, l'installazione, la gestione dei moduli, l'utilizzo delle *patch* e tutto quanto attiene la manutenzione dello stesso.

5.2.1 Le versioni del kernel

Uno dei primi problemi che ci si trova ad affrontare nella gestione del kernel è quello della scelta di quale versione usare. Nella maggior parte dei casi il kernel viene installato dalla propria distribuzione durante l'installazione, e molti, non avendo necessità specifiche (ad esempio la mancanza di supporto per un qualche dispositivo) evitano di installarne un altro.

Le esigenze che portano all'installazione di un nuovo kernel sono varie, la principale comunque resta quella di ottimizzare il kernel per renderlo più adatto alla propria configurazione hardware. Molti kernel di installazione infatti sono compilati con un supporto generico (per tipo di processore o per il chipset della piastra madre) per poter essere impiegati su qualunque PC, pertanto può essere utile ricompilarli per eliminare il supporto di funzionalità superflue non disponibili e attivare quello per la versione specifica del proprio hardware. Questa esigenza è sempre meno comune in quanto ormai tutte le distribuzioni forniscono versioni del kernel compilate specificatamente per i vari tipi di processore.

In questo caso si hanno due scelte, si può ricompilare il kernel della propria distribuzione (in genere tutte forniscono i relativi sorgenti), o utilizzare un kernel *ufficiale*, vale ad dire il kernel pubblicato su `http://www.kernel.org` a cura dal *mantainer* ufficiale (lo stesso Linus Torvalds o chi lui ha delegato al compito). In genere infatti le varie distribuzioni installano una propria versione del kernel, modificata applicando vari *patch* (vedi sez. 5.2.2) che si ritiene migliorino le prestazioni o la stabilità ed aggiungono funzionalità reputate rilevanti, ma non ancora incluse nel kernel ufficiale.

Qualora si scelga il kernel della propria distribuzione c'è solo da procurarsi i relativi sorgenti, i file di configurazione e provvedere alla ricompilazione secondo le istruzioni di sez. 5.2.3. Se invece si vuole installare un kernel ufficiale, ad esempio per avere le funzionalità aggiunte nello sviluppo effettuato nel frattempo, occorre scegliere una versione adeguata. La scelta della versione di kernel da utilizzare è in linea generale abbastanza semplice: occorre prendere l'ultima versione stabile. Le politiche di rilascio sono però cambiate notevolmente, ed anche se per stabilire di quale versione si tratta basta andare sul sito ufficiale del kernel, la logica che sta dietro la numerazione delle versioni è stata modificata profondamente varie volte.

Fino al kernel 2.6.15 versioni ufficiali erano caratterizzate da tre numeri separati da punti. Il primo numero esprime la *major version*, un numero di versione cambiato solo in caso di fondamentali modifiche strutturali dell'infrastruttura, cosa avvenuta finora una sola volta, nel passaggio dei formati dei binari dall'*a.out* all'ELF. La *major version* è sempre rimasta la 2 fino allo scadere dei venti anni di sviluppo del kernel, quando Linus ha deciso di passare alla versione 3 cambiando anche il significato delle cifre successive, anche se la versione 3.0 non ha introdotto nessuna modifica significativa rispetto alla 2.6.39.

Il secondo numero esprimeva il cosiddetto *patchlevel*, ed indica la serie di sviluppo. Questo fino alla serie 2.4.x era quello che veniva cambiato periodicamente in base ai cicli di sviluppo, si sono avute cioè le serie 2.0.x, 2.2.x, 2,4.x, fino alla 2.6.x. Prima della serie 2.6.x la convenzione scelta dagli sviluppatori era che un numero pari indicava una versione *stabile*, mentre un numero dispari indicava la versione *sperimentale*, di sviluppo, in cui venivano introdotte tutte le nuove funzionalità e le modifiche infrastrutturali che avrebbero portato alla successiva versione stabile.

Per esempio i kernel della serie 2.4.x erano i kernel stabili, usati per le macchine in produzione, lo sviluppo dei quali era centrato esclusivamente alla eliminazione dei bug e alla stabilizzazione del sistema, mentre i kernel della serie 2.5.x erano i kernel sperimentali, nei quali venivano introdotte le nuove funzionalità, riscritte parti che non si consideravano soddisfacenti, ecc.

L'ultimo numero di versione, detto *sublevel* veniva poi utilizzato come numero progressivo per identificare i vari rilasci successivi all'interno di una serie, nella direzione della stabilizzazione per le serie pari, nella direzione delle nuove funzionalità ed infrastrutture per quelle di sviluppo. Ogni serie stabile aveva un suo *mantainer* che coordinava lo sviluppo ed il rilascio delle nuove versioni mentre quello delle versioni di sviluppo è sempre stato lo stesso Linus Torvalds.

Con il rilascio della serie 2.6.x questo modello è stato totalmente cambiato; in precedenza infatti per un certo periodo tutti gli sforzi venivano concentrati nella stabilizzazione della nuova versione stabile, per poi dar vita ad una nuova versione di sviluppo. Accadeva spesso però che le distribuzioni, non volendo attendere i tempi lunghi del rilascio di una nuova versione stabile, introducessero nuove funzionalità o eseguissero il *backporting* di codice dalla versione di sviluppo a quella stabile, introducendo delle differenze significative fra le loro versioni e quella ufficiale.

Questo, insieme al fatto che diventava sempre meno sostenibile mantenere i kernel stabili rinunciando ai grandi miglioramenti introdotti in fase di sviluppo, è stato uno dei motivi che ha spinto al cambiamento di politica, diventato ufficiale nel novembre 2004 con il rilascio del kernel 2.6.9. La nuova politica non prevede la creazione di nuove serie di sviluppo, ma il rilascio di versioni successive nella stessa serie, integrando anche cambiamenti infrastrutturali significativi fra una versione stabile e la seguente.

Questo ha ovviamente comportato delle difficoltà sul piano della stabilità, dato che spesso emergevano errori sulla versione stabile in una fase successiva al suo rilascio, con il problema di come risolverli nel periodo di tempo, non sempre brevissimo, necessario per arrivare alla

versione stabile successiva, lo sviluppo della quale inizia sempre con un paio di settimane in cui si inseriscono tutte le funzionalità ed i cambiamenti che si ritengono maturi per l'ingresso nel kernel stabile, cui segue un periodo di stabilizzazione, che in genere dura al più qualche mese.

Per questo motivo è stato creato un team incaricato di fornire delle ulteriori sotto-versioni di *manutenzione* dei kernel stabili rilasciati, introducendo un quarto livello di numerazione. In queste sotto-versioni vengono inserite solo delle modifiche elementari o urgenti, come la chiusura di problemi di sicurezza e piccole correzioni per malfunzionamenti ed errori. La loro creazione e manutenzione inoltre normalmente viene terminata in corrispondenza del rilascio delle versioni stabili successive, ma in alcuni casi è stato deciso di mantenere alcune specifiche versioni per un tempo maggiore, con una sorta di supporto esteso.

Pertanto si sono avute situazioni in cui è avvenuto in contemporanea il rilascio di una nuova versione stabile (la 2.6.20) e quello di una versione di stabilizzazione (la 2.6.19.3) e la successiva cessazione della serie 2.6.19 dopo un breve tempo, o situazioni come quella del kernel 2.6.32 che è stato mantenuto per più di un anno, fornendo la base per molte distribuzioni.

Infine, dato che per questo cambiamento del modello di sviluppo la seconda cifra del numero di versione aveva completamente perso il significato che aveva in precedenza (restando fermi per anni con il 2.6), con il passaggio alla versione 3.0 nel luglio 2011 è stato semplicemente deciso di abolirla, pertanto il numero usato per il vecchio *sublevel* che esprimeva le versioni rilasciate a cadenze regolari è diventato il *patchlevel* ed i numeri progressivi delle ulteriori correzioni di stabilizzazione che prima andavano sulla quarta cifra sono diventati il nuovo *sublevel*, tornando di nuovo ad un numero di versione ufficiale di tre cifre separate da punti.

Si tenga comunque presente che la precedente indicazione di utilizzare l'ultimo kernel della serie stabile non ha un valore assoluto. Esistono infatti anche delle buone ragioni, come limiti sulle risorse e nel supporto di periferiche (i nuovi kernel tendono a occupare più memoria e a non supportare più piattaforme hardware particolarmente "*datate*") che spingono a mantenere l'utilizzo di kernel di serie precedenti, a partire dalla 2.0.x fino alla 2.4.x, che per questo sono tutt'ora mantenute, sia pure senza nessuno sviluppo, ed al solo livello della correzione dei pochi errori restanti.

5.2.2 Sorgenti e *patch*

Una volta scelta la versione del kernel da utilizzare, il passo successivo è quello di scaricare i sorgenti e ricompilarli. Come accennato il sito per la distribuzione delle versioni ufficiali è `http://www.kernel.org`, che in genere ha molto carico, per cui si consiglia l'uso di uno dei vari mirror italiani disponibili, la cui lista è segnalata sulla stessa pagina.

I sorgenti vengono distribuiti nella directory `/pub/linux` (vi si accede sia in FTP che in HTTP) e sono disponibili in tre forme,[20] le prime due sono degli archivi completi in formato `tar` compressi o con `bzip2` o con `gzip`, il cui nome sarà qualcosa del tipo `linux-2.6.19.tar.bz2` o `linux-2.6.19.tar.gz` (il primo è più compresso e si scarica più velocemente, ma si tratta comunque di varie decine di megabyte) oppure attraverso dei *patch* che permettono di passare da una versione precedente alla successiva, in modo che sia possibile evitare di riscaricare da capo l'archivio completo tutte le volte. Così ad esempio, una volta che si abbiano i sorgenti del kernel 2.6.18, si potrà passare al 2.6.19 scaricando soltanto il file `patch-2.6.19.gz` (di norma

[20] in realtà esiste anche una forma di distribuzione tramite il protocollo `rsync`, che permette di ridurre la quantità di dati da scaricare, ed attraverso il programma di gestione dei sorgenti `git`.

viene compresso anche questo). Così diventa possibile aggiornare alla versione successiva senza dover effettuare download di grosse dimensioni.

Una volta scaricati gli archivi si dovranno scompattare questi ultimi con `tar` col procedimento lo stesso illustrato in sez. 4.2.1 per l'installazione dei sorgenti, ottenendo una corrispondente directory `linux-2.6.xx` nella directory corrente. In genere si tende a mettere detti sorgenti in `/usr/src` ma nella procedura di compilazione ed installazione niente obbliga a questa scelta, se non il voler lasciare traccia delle versioni che si sono compilate in un posto noto.

Un discorso diverso va fatto qualora si vogliano utilizzare i *patch* per eseguire un aggiornamento di una precedente installazione dei sorgenti. Capire questa procedura è importante in quanto non riguarda soltanto il passaggio da una versione di kernel alla successiva, ma anche l'applicazione di eventuali *patch* relativi all'installazione di funzionalità aggiuntive che non sono ancora state incluse nei sorgenti del kernel ufficiale, ma delle quali si voglia introdurre il supporto. Esistono infatti molte estensioni rispetto al kernel cosiddetto "*vanilla*", anche di grande interesse, ma ancora sperimentali o molto specialistiche, che vengono mantenute al di fuori dello sviluppo del kernel ufficiale.

Per questo occorre capire cos'è un "*patch*". Il nome deriva proprio dall'uso più comune, che è quello che li vedeva rilasciati per mettere delle "*pezze*" a programmi con errori o problemi di sicurezza, si tratta semplicemente di un file il cui contenuto è definito dalle differenze di due file, in genere dei sorgenti di programmi, ma la cosa vale per qualunque file di testo, che viene prodotto con il comando `diff`. In questo modo è possibile ottenere quali righe sono cambiate da una versione all'altra e salvare su un file le sole differenze. Si può così passare da una versione di un programma alla successiva trasmettendo solo le differenze nel relativo sorgente.

Il comando `diff` può inoltre essere eseguito ricorsivamente su due intere directory, registrando le differenze sia per quanto riguarda i singoli file che esse contengono, che per l'aggiunta o la rimozione di alcuni di essi. Si può poi salvare il tutto su unico file, ed è questo che ad esempio viene distribuito come patch per passare da una versione stabile del kernel alla successiva; un esempio del contenuto di un *patch* è il seguente:

```
--- linux-2.4.25/arch/i386/defconfig    2004-02-18 05:36:30.000000000 -0800
+++ linux-2.4.26/arch/i386/defconfig    2004-04-14 06:05:25.000000000 -0700
@@ -2,7 +2,6 @@
 # Automatically generated make config: don't edit
 #
 CONFIG_X86=y
-CONFIG_ISA=y
 # CONFIG_SBUS is not set
 CONFIG_UID16=y

@@ -31,12 +30,14 @@
 # CONFIG_MPENTIUM4 is not set
 # CONFIG_MK6 is not set
 # CONFIG_MK7 is not set
+# CONFIG_MK8 is not set
 # CONFIG_MELAN is not set
 # CONFIG_MCRUSOE is not set
 # CONFIG_MWINCHIPC6 is not set
 # CONFIG_MWINCHIP2 is not set
 # CONFIG_MWINCHIP3D is not set
 # CONFIG_MCYRIXIII is not set
```

```
+# CONFIG_MVIAC3_2 is not set
 CONFIG_X86_WP_WORKS_OK=y
 CONFIG_X86_INVLPG=y
 CONFIG_X86_CMPXCHG=y
...
```

dove la prima riga indica qual'è file originale da cui si è partiti, mentre la seconda la nuova versione. Si noti che si tratta di un pathname relativo, che ha come origine la directory in cui si trovano i due diversi alberi quando è stato eseguito il `diff`. Le righe che iniziano per `@@` indicano a quale riga nei due file fanno riferimento i dati riportati di seguito, nel caso 7 righe del primo file a partire dalla 12, che diventano 6 del secondo a partire sempre dalla stessa riga. Le differenze sono mostrate apponendo un "`+`" alle righe aggiunte nel secondo file ed un "`-`" a quelle tolte.

Con il comando `patch` si può invece compiere l'operazione inversa rispetto a `diff`, e cioè applicare ad un certo file le differenze ottenute con il metodo precedente, in modo da convertirlo nella nuova versione. La cosa può essere effettuata anche ricorsivamente, su un intero albero di file e directory. Così chi dispone sia dei nuovi sorgenti del kernel 2.4.26 che di quelli della precedente versione 2.4.25 potrà generare un *patch* con tutte le differenze.

Questo applicato da una persona ai sorgenti del kernel 2.4.25 li trasformerà in quelli del 2.4.26. L'utilità dei *patch* è che in questo modo anche chi cura la manutenzione di ulteriori funzionalità non presenti nei kernel ufficiali potrà limitarsi a distribuire il *patch* che contiene le relative aggiunte e modifiche, così che un utente possa, se lo desidera, applicarle senza dover scaricare tutto un nuovo albero dei sorgenti.

Il meccanismo è del tutto generale, ed inoltre il comando `patch` è sufficientemente intelligente da essere in grado di applicare su uno stesso albero anche più *patch* distinti in successione, fintanto che questi non vadano ad operare esattamente sulle stesse righe degli stessi file eseguendo modifiche incompatibili fra di loro. Così diventa possibile, applicando in successione più *patch*, inserire nel kernel più funzionalità aggiuntive, fintanto che queste non interferiscono fra loro. Allo stesso modo è possibile passare da una versione di kernel ad un'altra che non sia la successiva.

Come accennato il comando per applicare un *patch* è appunto `patch`. Nel caso più semplice, in cui si deve operare su un singolo file, la sua sintassi è immediata, e si può eseguire il comando con un qualcosa del tipo:

```
patch original patch.diff
```

nel qual caso, a meno che non si sia specificata l'opzione `-b` (o `--backup`) per richiedere un backup, la nuova versione prenderà il posto dell'originale.

Quando però si ha a che fare con un *patch* che coinvolge più file (come quelli che si applicano ad un albero di sorgenti) i nomi dei file cui esso va applicato sono riportati nel file stesso, e non devono quindi essere specificati; inoltre in questo caso il comando legge il contenuto del *patch* dallo standard input, per cui occorre usare una redirezione.

Per capire il funzionamento del comando conviene rifarsi all'esempio di *patch* mostrato in precedenza: il comando ricerca (a partire dalla directory corrente) il file che considera la vecchia versione e cerca di applicarvi le differenze. Il problema che ci si trova di fronte è che normalmente si ha a disposizione solo la versione di partenza e non quella di arrivo, ad esempio si potranno avere i sorgenti del kernel 2.4.25 nella directory `linux-2.4.25` ma non ci sarà la directory `linux-2.4.26`. Per questo motivo di solito bisogna dire al comando da quale livello si vuole partire per applicare il *patch*, usando l'opzione `-p`.

Il livello 0 usa semplicemente quanto specificato nel *patch* stesso, ma nel caso dell'esempio di *patch* illustrato in precedenza questo non può funzionare, in quanto il file di destinazione non esiste. Se però ci si pone direttamente dentro la directory `linux-2.4.25`, ignorando il primo livello di directory, tutti i pathname relativi verranno risolti senza problemi. Per questo quello che si fa per applicare un *patch* sui sorgenti del kernel è qualcosa del tipo:

```
cd /usr/src/linux
patch -p1 < /path/to/patch/patch.diff
```

Si tenga presente che se non si specifica un livello, il default di `patch` è di utilizzare solo il nome del file, ignorando le directory presenti nei pathname relativi in esso contenuti, per cui in genere l'applicazione fallirà. Se il comando non riesce ad applicare un *patch*, ad esempio perché se ne è già applicato uno incompatibile, o si è sbagliato file, genererà automaticamente dei file terminanti in `.rej` che contengono le modifiche che è stato impossibile effettuare.

Inoltre `patch` è in grado di rilevare il caso in cui si è prodotto il *patch* invertendo le versioni, nel qual caso avvisa richiedendo il permesso di applicare il *patch* alla rovescia. Questo può essere richiesto esplicitamente con l'opzione `-R` (o `--reverse`). Si tenga presente però che se si tenta di applicare uno stesso *patch* due volte ci si troverà proprio in questa situazione e proseguire nell'applicazione non sarebbe corretto, per questo esiste l'opzione `-N` (o `--forward`) che indica di ignorare i *patch* che sembrano invertiti o già applicati.

Il comando `patch` prende molte altre opzioni ed è in grado di utilizzare vari formati per i *patch* ed anche di interagire direttamente con vari programmi per il controllo di versione per identificare quali sono i file su cui operare. Per tutti i dettagli sul funzionamento del comando e sul significato delle opzioni si può al solito fare riferimento alla pagina di manuale, accessibile con `man patch`.

5.2.3 La ricompilazione del kernel

Una volta che si sono scompattati i sorgenti ed applicati gli eventuali *patch* ritenuti opportuni si può passare alla compilazione del kernel. Questa, come per la maggior parte dei pacchetti che si installano dai sorgenti, viene eseguita tramite il comando `make`, ma nel caso non viene utilizzata la procedura illustrata in sez. 4.2.1 perché nel caso del kernel non esiste uno script di configurazione e tutto viene gestito attraverso una procedura di costruzione dedicata, creata dagli stessi sviluppatori. Pertanto tutto la procedura è controllata dal `Makefile` principale presente nella base della directory dei sorgenti, e le varie operazioni sono compiute invocando gli opportuni *target* del comando `make` (si ricordi quanto detto in sez. 4.2.1).

Una delle caratteristiche peculiari di Linux (torneremo sull'argomento in dettaglio anche in sez. 5.2.5) è quella di essere *modulare*. A differenza cioè degli altri sistemi unix-like in cui il kernel è un unico programma *monolitico*, caricato in memoria all'avvio del sistema ed in cui devono essere inserite tutte le funzionalità che si vogliono usare, Linux può partire con un kernel contenente le sole funzionalità di base e poi caricare da disco in maniera dinamica delle ulteriori sezioni di codice, dette moduli, che aggiungono funzionalità aggiuntive o il supporto di certi dispositivi solo quando servono.

Questa è una delle caratteristiche più rilevanti di Linux, che gli permette una flessibilità di utilizzo che i kernel monolitici tradizionali non hanno. É possibile infatti modularizzare lo sviluppo del kernel separandone le funzionalità, ed evitare di mantenere permanentemente in

memoria parti di codice che sono utilizzate solo per limitati periodi di tempo (ad esempio il codice per accedere a CDROM o floppy occupa inutilmente memoria se non li si stanno utilizzando), indicare opzioni specifiche per la gestione di un dispositivo in fase di caricamento, o modificarle senza bisogno di un riavvio (basta rimuovere il modulo e ricaricarlo con le nuove opzioni).

L'uso dei moduli ha pertanto una grande rilevanza e deve essere pianificato accuratamente in fase di compilazione e configurazione. Un primo aspetto dell'uso dei moduli è che quando si usano diverse versioni del kernel devono essere usate anche diverse versioni dei moduli. Ciò comporta che ogni kernel deve avere la sua versione dei moduli, che sono identificati, come quest'ultimo, per la relativa versione, quella che viene mostrata dal comando `uname -r`.[21] Sorge allora un problema quando si vogliono ottenere due (o più) kernel diversi a partire dagli stessi sorgenti, dato che in questo caso la versione sarà la stessa.

Per risolvere questo problema è allora possibile definire una versione "*personalizzata*". La versione del kernel è indicata dai sorgenti, ed è codificata nelle prime righe del `Makefile` principale, che per i kernel ufficiali sono nella forma:

```
VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 27
EXTRAVERSION =

KERNELRELEASE=$(VERSION).$(PATCHLEVEL).$(SUBLEVEL)$(EXTRAVERSION)
```

e come si vede è qui che viene definita la variabile `KERNELRELEASE` che poi sarà usata in tutto il resto della procedura. Si noti allora la presenza, appositamente predisposta, della variabile `EXTRAVERSION`, che serve appunto a specificare un ulteriore identificativo di versione, che permetta di tenere separati kernel diversi (ad esempio per le opzioni di compilazione che si sono scelte) ottenuti a partire dagli stessi sorgenti.

Nelle attuali versioni ufficiali `EXTRAVERSION` resta sempre non definita, è cura di chi esegue una ricompilazione definirla adeguatamente, evitando di usare caratteri speciali, come lo spazio, che possono essere interpretati negli script (il caso più comune è aggiungere un "`-qualcosa`"). Questo però non è vero per le release intermedie degli aggiornamenti delle serie 2.6.x.y e per i sorgenti mantenuti indipendentemente da altri sviluppatori, che hanno usato appunto questa variabile per marcare le loro versioni.

In genere questa è l'unica modifica che può essere necessario fare a mano a questo file,[22] tutte le altre configurazioni sono gestite in maniera indipendente, attraverso la modalità che vedremo più avanti, ma a partire dal kernel 2.6.x è prevista una opzione per impostarla direttamente anche dall'interno del programma di configurazione e non è più necessario modificare direttamente il `Makefile`.

Le altre eventuali modifiche che si possono voler fare al `Makefile`, molto specialistiche, riguardano la variabile `CROSS_COMPILE`, che può essere usata per compilare kernel per una architettura diversa dalla propria (ad esempio un kernel per PowerPC su una macchina Intel), e le opzioni per le ottimizzazioni del `gcc`, che è meglio lasciar stare al valore di default, che è sicuro, a meno di non sapere esattamente quello che si sta facendo, o essere in vena di sperimentazione.

[21] in realtà come vedremo in sez. 5.2.5, i moduli sono identificati soprattutto per la directory in cui sono mantenuti, che è `/lib/modules/'uname -r'`.

[22] anche se il `kernel-package` di Debian fornisce il comando `make-kpkg` che è in grado di farla automaticamente.

Un secondo aspetto dell'uso dei moduli che occorre tener presente è che per poterli utilizzare occorre anzitutto poter caricare in memoria il loro codice; il che significa che si deve essere in grado di usare i relativi file oggetto,[23] dato che questi non sono altro che codice, come quello dei programmi ordinari, anche se un po' particolare. A partire dalla serie 2.6.x, per distinguerli dai file *oggetto* dei normali programmi, è stata adottata l'estensione `.ko` al posto di `.o`, ma si ricordi che l'estensione in Unix è solo una convenzione, nel caso conta solo il contenuto.

L'uso dei moduli comporta però che le funzionalità necessarie ad accedere al supporto su cui si trovano questi file non possono essere ottenute in forma modulare e dovranno essere inserite all'interno del kernel in maniera *monolitica*. Per capire quali sono queste funzionalità occorre ricordare quali sono i due compiti di base eseguiti dal kernel all'avvio: montare la directory radice ed eseguire il programma di inizializzazione di sistema (in genere `/sbin/init`).

Per il primo compito occorre il supporto per accedere al dispositivo su cui si trova la radice e quello per il relativo filesystem, per il secondo il supporto per l'uso del formato binario di esecuzione dei programmi. Quanto necessario a svolgere questi due compiti, anche se modularizzabile, dovrà comunque essere inserito permanentemente nel kernel, pena il fallimento dell'avvio con un *kernel panic*, vale a dire con un crash fatale del kernel, che comporta il blocco completo della macchina, e che può avvenire solo per errori fatali durante l'esecuzione dello stesso (in caso di bug particolarmente gravi, molto rari, o di problemi hardware, assai meno rari), oppure come nel caso in questione, se mancano delle componenti essenziali per l'avvio del sistema.

A parte le eventuali modifiche del `Makefile` per modificare la `EXTRAVERSION`, il primo passo per la compilazione del kernel è quello della configurazione, in cui si scelgono quali funzionalità attivare e quali no, quali mettere direttamente dentro il kernel, e quali utilizzare come moduli. Una volta che si sia specificato quanto voluto, il kernel verrà costruito di conseguenza.

Tutto questo viene fatto, dal punto di vista della compilazione e della costruzione del kernel, tramite il contenuto del file `.config`, sempre nella directory base dei sorgenti, dove sono memorizzate tutte le opzioni di configurazione. Un estratto del contenuto del file è il seguente:

`.config`

```
...
#
# Automatically generated make config: don't edit
#
# CONFIG_UID16 is not set
# CONFIG_RWSEM_GENERIC_SPINLOCK is not set
CONFIG_RWSEM_XCHGADD_ALGORITHM=y
CONFIG_HAVE_DEC_LOCK=y
...
```

Le opzioni di configurazione sono tutte dichiarate come variabili nella forma `CONFIG_XXX`, quelle attivate non sono commentate ed assegnate al relativo valore, alcune indicano dei valori generici, come il tipo di processore o la codifica NLS (il *Native Language Support*), cioè la codifica dei caratteri delle stringhe, come `iso8859-1` o `utf-8`, usata di default, ma la maggior

[23] un file oggetto, in genere identificato dall'estensione `.o`, è un file che contiene il codice compilato di una o più funzioni, in cui però gli indirizzi non sono stati assegnati; in questo modo, attraverso un procedimento successivo detto *collegamento* (in inglese *linking*) si possono unire insieme più funzioni per dar luogo a quello che poi andrà a costituire un eseguibile; questo è anche quello che si fa quando si produce l'immagine del kernel, e l'uso dei moduli consente di ripetere il procedimento sul codice del kernel in esecuzione.

parte possono avere come valori possibili soltanto "y", che richiede l'inclusione nel kernel o la semplice attivazione della stessa, o "m", che richiede l'utilizzo come modulo, e che ovviamente è disponibile soltanto qualora l'opzione faccia riferimento ad una funzionalità che può essere modularizzata.

Come scritto in testa all'estratto illustrato in precedenza, il file `.config` non deve essere scritto a mano, ma opportunamente generato, dato che, a meno di non sapere esattamente quello che si sta facendo, si rischia di attivare opzioni incompatibili fra di loro o inconsistenti. Per questo la generazione di questo file viene eseguita invocando `make` con uno dei *target* di configurazione.

Il primo *target* di configurazione possibile è `config`; questo avvia uno script di shell che effettua la configurazione chiedendo di immettere sul terminale uno per uno i valori da assegnare alle varie opzioni che si vogliono attivare. Ovviamente eviterà di eseguire ulteriori domande qualora non si attivi una opzione che le prevede, ma il procedimento è comunque molto scomodo in quanto non esiste un meccanismo per correggere una impostazione una volta che si sia fatto un errore, per cui occorre ricominciare da capo.

Pertanto è oggi praticamente in disuso, a parte per una sua versione modificata, invocabile con il *target* `oldconfig`, che si limita a rileggere e riprocessare il file di configurazione precedente ricavando da questo, invece che dalle nostre risposte sul terminale, le risposte alle domande. Questo può risultare utile qualora si siano effettuate modifiche a mano del file `.config` e si voglia essere sicuri di ottenere un file di configurazione coerente.

Gli altri *target* di configurazione più usati sono `menuconfig` e `xconfig` che attivano invece due interfacce utente, testuale la prima e grafica la seconda, con finestre e menu che permettono di selezionare interattivamente le varie opzioni ed effettuare le relative scelte in maniera casuale, senza serializzare le domande. Le due interfacce, a parte l'apparenza, sono sostanzialmente equivalenti, per cui tratteremo solo la prima. A partire dal kernel 2.6.x le interfacce grafiche sono diventate 2; la prima, sempre accessibile con `make xconfig` è basata sulle librerie QT, la seconda, accessibile con `make gconfig`, è basata sulle librerie GTK.

Eseguendo `make menuconfig` nella directory dei sorgenti del kernel si otterrà la pagina di avvio del programma di configurazione, mostrata in fig. 5.2. In genere il programma viene compilato la prima volta che si esegue il relativo *target*, questo talvolta fallisce in quanto per la compilazione necessitano le librerie *ncurses* su cui è basata l'interfaccia a finestre. Di norma queste vengono installate, ma non altrettanto avviene per i file di dichiarazione necessari alla compilazione (si ricordi quanto detto in sez. 4.2.1), nel qual caso andrà installato il relativo pacchetto di sviluppo[24] e quelli delle *glibc*, qualora anch'essi fossero assenti.

La finestra di avvio riporta nella prima riga in alto la versione del kernel, se si è modificata la `EXTRAVERSION` questa dovrà comparire. Subito sotto c'è il titolo della sezione in cui ci si trova (nel caso è il menu principale), seguito da un breve riassunto dei principali comandi disponibili. Le frecce verticali permettono di spostarsi nella finestra centrale che contiene le varie sezioni in cui sono state suddivise le opzioni di configurazione. Nella parte bassa ci sono le tre opzioni principali che permettono di selezionare una opzione, uscire dalla finestra corrente e ottenere una finestra di aiuto (contestuale all'opzione selezionata), che possono essere cambiate con le frecce orizzontali.

Dal menu principale è possibile selezionare una sezione premendo invio (a meno di non aver cambiato l'opzione di selezione), e questo ci porterà nella finestra di configurazione delle relative

[24]le librerie *ncurses* sono presenti in tutte le distribuzioni, per cui è sempre il caso di usare i relativi pacchetti, per Debian sono `ncurses` e `libncurses-dev`.

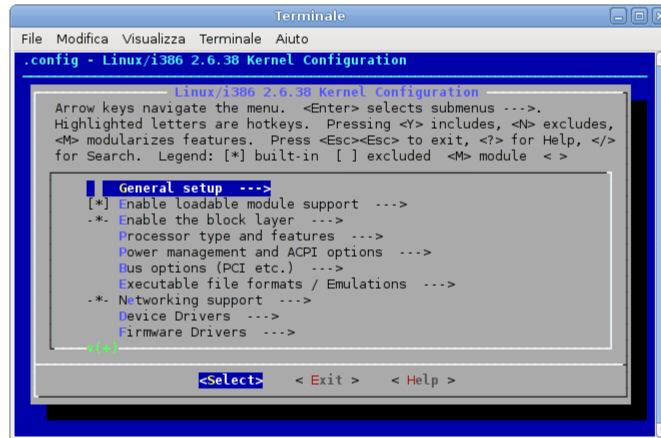


Figura 5.2: Schermata di avvio della configurazione del kernel con `make menuconfig`.

opzioni, un esempio della quale è mostrato in fig. 5.3. I valori delle opzioni sono riportati all'inizio di ogni riga, quelli indicati fra parentesi tonde sono per le opzioni che richiedono un valore generico, che può essere selezionato da un menu a tendina o inserito da una finestra di immissione che si attiva quando l'opzione viene selezionata.

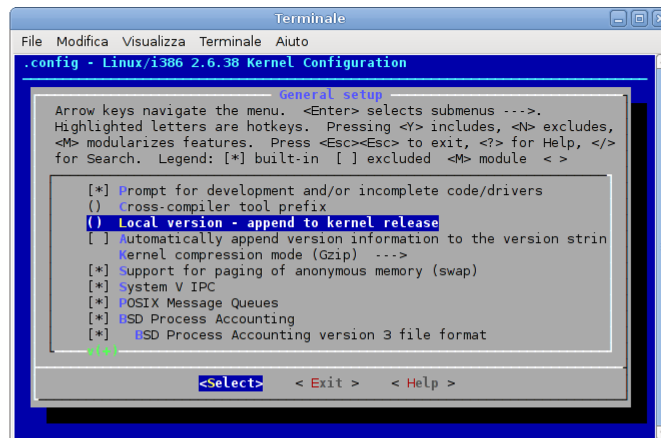


Figura 5.3: Schermata di configurazione del kernel con `make menuconfig`.

I valori fra parentesi quadre indicano le opzioni per le quali è possibile solo scegliere fra l'attivazione o meno e la scelta può essere fatta premendo il tasto "`y`" per attivare e "`n`" per disattivare, premendo la barra si può ciclare fra le due opzioni. Un asterisco indica che la funzionalità è attivata, uno spazio vuoto che non è attivata. Si tenga presente che se si tratta del supporto per funzionalità specifiche del kernel questo implica che il relativo codice sarà incluso monoliticamente, molte di queste opzioni però servono anche per attivare ulteriori configurazioni

o specificare caratteristiche di un'altra opzione (che può anche essere modulare), nel qual caso le successive descrizioni appariranno indentate.

I valori fra parentesi angolari indicano invece le opzioni relative a funzionalità che possono essere anche modularizzate; rispetto alle precedenti possono presentare anche il valore "`M`" (attivabile direttamente premendo "`m`" o ciclando fra i valori possibili con la barra) che indica che si è optato per la creazione del relativo modulo, se invece si ha un "`*`" la funzionalità sarà inserita direttamente nel kernel.

Infine per alcune sezioni sono presenti delle ulteriori sottosezioni (sono indicate dalla assenza del valore delle opzioni e dal fatto che terminano con una freccia, come nel menu principale). Una volta che si sono effettuate le proprie scelte selezionando la voce di uscita si può tornare al menu precedente, se si è già nel menu principale invece si uscirà direttamente dal programma di configurazione, e comparirà una finestra che richiede se si vuole che le nuove configurazioni siano salvate sovrascrivendo un nuovo `.config`, o scartate.

Due opzioni specifiche per il menu principale sono poi quelle disponibili separatamente in fondo allo stesso, che permettono di salvare i valori della configurazione su, o caricarli da, un file specificabile dalla solita riga di immissione dei dati. Questo può comunque essere ottenuto con delle semplici copie del file `.config`.

Data la quantità (molte centinaia) di opzioni disponibili non è possibile commentarle tutte, pertanto ci limiteremo ad una descrizione sommaria del contenuto delle varie sezioni del menu principale. Queste sono state completamente riorganizzate nel passaggio dal kernel 2.4.x al 2.6.x, ed inoltre nel corso dello sviluppo ne state aggiunte delle altre; alla stesura di questo testo[25] erano disponibili:

General setup

Questa è la sezione dove si attivano le opzioni di configurazione generali; qui si può impostare la *extraversion* del kernel (con "`Local version - append to kernel release`") e le opzioni che consentono la visualizzazione di altre opzioni specialistiche come "`Prompt for development and/or incomplete code/drivers`" che permette di attivare tutte le opzioni che sono classificate come sperimentali o "`Configure standard kernel features (for small systems)`" che permette di eseguire ulteriori configurazioni per i sistemi con poche risorse. In particolare è sempre opportuno abilitare "`Prompt for development and/or incomplete code/drivers`" prima di iniziare una configurazione in quanto buona parte delle opzioni sperimentali sono ampiamente utilizzate e perfettamente funzionanti; le opzioni "*pericolose*" vengono adeguatamente segnalate nelle relative descrizioni.

Inoltre sono presenti in questa sezione le opzioni per il supporto per alcune funzionalità fornite dal kernel, come i meccanismi di intercomunicazione fra processi, il controllo e la suddivisione delle risorse,[26] e vari supporti per tecnologie di debug, profilazione e controllo delle prestazioni. Infine vanno esplicitamente ricordate due opzioni molto importanti per il funzionamento del sistema: il supporto per il *ramdisk* iniziale (vedi sez. 5.2.4) con l'opzione "`Initial RAM filesystem and RAM disk`

[25] la revisione è dell'Aprile 2013, e fa riferimento al kernel 3.1.4.

[26] come i *control group* (`Control Group support`) e varie funzionalità (`Namespace support`) utilizzate per il supporto ai cosiddetti *container*, una tecnologia che consente di eseguire macchine virtuali all'interno di *contenitori* forniti dal sistema, senza dover eseguire una virtualizzazione dell'hardware.

`(initramfs/initrd) support`” e per la *swap* (vedi sez. 5.1.7) con l'opzione “`Support for paging of anonymous memory (swap)`”.

Loadable module support

In questa sezione si attivano le opzioni per abilitare la gestione dei moduli ed il relativo supporto nel kernel. In generale si possono attivare tutte, mentre si possono disattivare quelle che introducono dei controlli di versione per i moduli, in particolare quella relativa ai sorgenti. In generale è utile mantenere attiva quella marcata “`Module versioning support`” per evitare che vengano caricati per errore moduli compilati per un'altra versione del kernel non compatibile; questo può comportare problemi qualora si vogliano utilizzare moduli compilati a parte o distribuiti in forma binaria. Nel 2.4 il meccanismo funzionava aggiungendo una *checksum* a tutti i nomi dei simboli del kernel, cosicché le relative funzioni possono essere chiamate solo all'interno dello stesso kernel, nel 2.6 questa informazione viene mantenuta a parte e può essere ignorata.

Enable the block layer

Questa sezione contiene le opzioni relative alla gestione generica dei dispositivi a blocchi che deve essere sempre inserita nel kernel perché il sistema possa funzionare (deve poter montare la radice). Qui si trovano in questa sezione anche le opzioni relative ai diversi *scheduler* (`IO schedulers`) usabili per ottimizzare l'I/O su disco. Il kernel è in grado di utilizzare diverse strategie per organizzare l'accesso a disco in maniera ottimale da parte di diversi processi, queste possono essere stabilite in fase di avvio ed il default può essere impostato con l'opzione “`Default I/O scheduler`”, presente in questa sezione.

Processor type and features

Questa sezione contiene le opzioni relative alla scelta del tipo di CPU presente, con le opzioni per il supporto di vari insiemi di istruzioni estese (“`MTRR (Memory Type Range Register) support`”), ai vari tipi di timer usati dal sistema, la frequenza dello stesso (“`Timer frequency`”). Di particolare importanza è poi l'opzione per il “`Symmetric multi-processing`” che consente l'uso di macchine multiprocessore, e quelle (“`Preemption Model`”) che controllano le modalità di gestione dello *scheduler* dei processi a seconda del tipo di utilizzo. Sempre qui vanno attivate una serie di opzioni relative alla gestione della memoria, come quelle per l'uso di grandi quantità di memoria (“`High Memory Support`”) quando si ha più di 1Gb su macchine a 32 bit o il “`Transparent Hugepage Support`” per l'uso automatico di pagine di memoria di maggiori dimensioni che riducano la frammentazione della stessa. Si trovano qui anche alcune opzioni di supporto per essere eseguito come ospite in una macchina virtualizzata (“`Paravirtualized guest support`”).

Power management options and ACPI options

Questa sezione contiene tutte le opzioni relative al supporto delle funzionalità di risparmio energetico e di gestione dell'alimentazione, spegnimento, sospensione su disco ed in RAM ecc. È qui che si configura il supporto per l'ACPI (“`Advanced Configuration and Power Interface`”) e per tutte le opzioni di controllo delle modalità di risparmio energetico del processore o della memoria.

Bus options (PCI etc.)
Questa sezione contiene tutte le opzioni di configurazione relative a vari bus hardware (vedi sez. 5.4) supportati da Linux, come PCI, PCMCIA, ecc. Sono di particolare importanza le opzioni relative al bus PCI che devono essere sempre inserite all'interno del kernel in quanto questo è il bus usato dalla quasi totalità dei computer odierni per accedere alle periferiche principali (dischi, scheda video, schede sonore, ecc.), ed è pertanto necessario per poter accedere ai dischi e montare la radice.

Executable file formats
Questa sezione contiene le opzioni relative al supporto per i formati binari dei file che il kernel è in grado di eseguire come programmi. Il supporto per il formato ELF deve essere sempre abilitato nel kernel (altrimenti sarà impossibile eseguire `/sbin/init`), gli altri possono essere abilitati come moduli.

Networking support
Questa sezione contiene le sottosezioni relative al supporto delle varie tecnologie di rete, wireless, infrarossi, *bluetooth*, CAN, ecc. per ciascuna delle quali occorre abilitare (anche come moduli) le funzionalità necessarie. Di particolare importanza è poi, raccolto nella sottosezione "`Networking options`", il supporto per i vari protocolli di rete ed ai diversi tipi di socket e le funzionalità relative al filtraggio dei pacchetti ("`Network packet filtering`") ed al routing avanzato; in questa sottosezione sono da attivare "`TCP/IP networking`", "`Packet socket`" e "`Unix domain sockets`", per gli ultimi due è possibile farlo anche in maniera modulare. Oltre a questi si può poi abilitare il supporto per tutta una serie di altri protocolli di rete o estensioni dei protocolli standard come IPSEC.

Device Drivers
Questa è la sezione più corposa e più importante in quanto contiene le opzioni di configurazione per tutti vari tipi di periferiche che si possono utilizzare con Linux; tratteremo separatamente in coda a questa panoramica sulle opzioni del menu principale le varie sottosezioni in cui essa è suddivisa.

Firmware Drivers
Questa sezione, relativamente recente, contiene le opzioni per il supporto di alcune interfacce di comunicazione con il *firmware* (il BIOS o alternative) della propria macchina.

File systems
Questa sezione contiene le opzioni per il supporto di un gran numero di diversi filesystem. Si deve essere sicuri di inserire nel kernel il supporto per il filesystem della radice, qualunque esso sia (i più usati sono i vari *extN*, *reiserfs* e XFS). Qui si può anche abilitare, da altrettante sotto-sezioni, il supporto per i filesystem di CDROM e DVD, e per i vari filesystem di Windows. É sempre opportuno abilitare il supporto per il filesystem `/proc` (che è usato da moltissimi programmi) nella sotto-sezione "`Pseudo filesystems`". Una sotto-sezione a parte ("`Network File Systems`") è dedicata ai filesystem di rete (NFS, SMB ed altri), dove può essere configurato il relativo supporto. Infine si trovano in questa sezione anche le opzioni relative alla gestione delle quote disco (vedi sez. 6.4), dell'*automounter* (vedi sez. 5.1.6), dei vari meccanismi

di notifica delle modifiche dei file principalmente ad uso dei programmi ad interfaccia grafica, ed il supporto per la realizzazione di filesystem in user-space (FUSE).

Kernel hacking
Questa sezione contiene le opzioni per la configurazione del supporto di una lunga serie di funzionalità di controllo utilizzate principalmente dagli sviluppatori per il debug del kernel. Può essere utile abilitare l'opzione "`Magic SysRq key`" che permette l'uso di particolari combinazioni di tasti (a partire appunto da *SysRq*) per tentare un recupero in estremo dei dati in caso di crash del kernel.

Security options
Questa sezione contiene le opzioni relative alle infrastrutture di sicurezza (come *SELinux*, *Tomoyo*, *AppArmor* o eventuali altri moduli che consentono di supportare diverse politiche di sicurezza) che possono essere attivate all'interno del kernel. É qui che si sceglie quale di questi moduli deve essere usato di default.

Cryptographic API
Questa sezione contiene le opzioni per il supporto di vari algoritmi crittografici all'interno del kernel, compreso l'eventuale supporto hardware per alcuni di questi. In genere viene utilizzato per supportare filesystem e protocolli di rete cifrati. Sono inserite in questa sezione anche le opzioni per abilitare gli algoritmi di compressione.

Virtualization
Questa contiene le opzioni di configurazione per il supporto di varie tecnologie di virtualizzazione dell'hardware con cui diventa possibile fare eseguire dal kernel un altro sistema operativo a velocità nativa, creando opportunamente una *macchina virtuale* sia in termini di emulazione del processore che di alcuni dispositivi.

Library routines
Questa sezione contiene le opzioni per la configurazioni di alcune funzioni di libreria usate dal kernel per lo più per calcoli di somme di controllo.

Il precedente elenco riporta solo un breve sommario dei contenuti delle varie sezioni in cui sono state suddivise le varie opzioni nel menu principale del programma di configurazione del kernel. Come accennato però la sezione "`Device Drivers`" ha una rilevanza particolare in quanto è al suo interno che si trovano la maggior parte delle opzioni di configurazione.

Detta sezione infatti è divisa in un gran numero di ulteriori sottosezioni, relative alle varie tipologie di dispositivi che il kernel è in grado di supportare; per questo motivo si è deciso di riportare di seguito anche una breve descrizione delle principali sottosezioni in essa contenute:

Generic Driver Options
Contiene le opzioni generali relative alla gestione dei driver, in particolare il supporto per il caricamento dei firmware sulle periferiche e la creazione di un contenuto minimo per `/dev` ad uso di emergenza, disponibile anche senza la presenza di *udev* (vedi sez. 5.4.5). Consentiva anche l'indicazione, ora deprecata, del programma da utilizzare per la gestione degli eventi generati dal kernel che nelle versioni recenti vengono gestiti da *udev*.

Memory Technology Devices (MTD) support
Questa sezione contiene le opzioni relative ai supporti di memoria opzionali come flash, memorie a stato solido ecc. Vengono utilizzate prevalentemente nei sistemi embedded, ancorché alcune di queste stanno trovando uso anche in sistemi ordinari dotati di dischi a stato solido.

Parallel port support
Questa sezione contiene le opzioni relative al supporto per la porta parallela, necessarie per poter utilizzare i dispositivi (ad esempio una stampante) ad essa collegati. Può essere completamente modulare ed è in disuso per la sempre minore diffusione di questa interfaccia.

Plug and Play support
Questa sezione contiene le opzioni per abilitare il supporto all'uso del *Plug and Play* per le schede che lo supportano (vedi sez. 5.4.1). Riguarda hardware ormai praticamente scomparso e può essere completamente modulare.

Block devices
Questa sezione contiene le opzioni per abilitare il supporto di una serie di dispositivi a blocchi (i floppy, vari *disk-array* e RAID hardware, i *ramdisk* e il *loopback*). A meno di non avere la radice su uno di questi dispositivi il supporto può essere modulare. A parte il supporto per i sempre meno comuni floppy (con "`Normal floppy disk support`") le funzionalità più usate sono quelle relative ai dischi in RAM ("`RAM block device support`") e al dispositivo di *loopback* ("`Loopback device support`") già visto in sez. 5.1.3), in particolare se si usa un meccanismo di avvio che utilizza un *ramdisk* un occorrerà inserire "`RAM block device support`" all'interno del kernel. Si configurano qui anche il dispositivo a blocchi virtuale ad uso di macchine virtualizzate ("`Virtio block driver`") ed il RAID via rete con DRBD ("`DRBD Distributed Replicated Block Device support`").

Misc devices
Questa sezione contiene le opzioni relative ad una miscellanea di dispositivi diversi, molti dei quali relativi al supporto di funzionalità presenti su specifiche macchine (in genere portatili); può essere completamente modulare.

ATA/ATAPI/MFM/RLL support
Questa sezione contiene le opzioni per il supporto del bus IDE/ATA tradizionale (detto talvolta anche *Parellel ATA*), e di tutti i relativi dispositivi (dischi, CDROM, ecc.). Questa sezione fa riferimento al supporto classico del bus IDE e non al nuovo supporto tramite *libata*, la cui configurazione si effettua nella successiva sezione "`Serial ATA and Parallel ATA drivers`" e nei kernel più recenti è deprecata. Qualora si utilizzi questa modalità di supporto e la radice sia su un disco IDE occorrerà abilitarla ed inserirla direttamente nel kernel, inoltre al suo interno occorrerà abilitare le due opzioni "`generic/default IDE chipset support`" e "`generic ATA/ATAPI disk support`", oltre al supporto specifico per il proprio chipset; tutto il resto può essere modulare. Si può lasciare attivo "`Generic PCI IDE Chipset Support`" che permette l'avvio con un supporto generico. Questo supporto è però in dismissione e la sezione è deprecata,

in quanto tutto quello che non è relativo ad hardware in disuso è già stato trasferito nella nuova sezione citata.

SCSI device support
Questa sezione contiene le opzioni per il supporto dei dispositivi SCSI (come dischi, CDROM, nastri), del relativo protocollo, e dei vari controller. L'opzione "`SCSI support`" deve essere abilitata anche se non si hanno dispositivi SCSI in quanto il protocollo viene usato da altri sistemi, come i programmi per la masterizzazione (che usano l'emulazione IDE-SCSI) e le chiavette e i dischi esterni USB o i dischi *Serial ATA* (che usano il protocollo SCSI per accedere ai dati). Se si ha la radice su un disco SCSI tutto tranne "`SCSI device support`", "`SCSI disk support`" ed il supporto per il proprio controller può essere modulare. Nella sottosezione "`SCSI low-level drivers`" si può poi selezionare il supporto per i vari tipi di controller.

Serial ATA and Parallel ATA drivers
Questa sezione contiene le opzioni per il supporto dei nuovi dispositivi *Serial ATA*, e, con le versioni più recenti del kernel, anche quello per alcuni dispositivi che usano l'interfaccia IDE/ATA tradizionale (il cosiddetto *Parallel ATA*); in quest'ultimo caso questo supporto va a sostituire quello della precedente sezione "`ATA/ATAPI/MFM/RLL support`". Se si ha la radice su questo tipo di dispositivi, oltre a quello dello specifico chipset della propria scheda madre, occorre anche abilitare il supporto SCSI per i dischi (le due opzioni "`SCSI device support`", "`SCSI disk support`" della precedente sezione), dato che in questo caso i dischi vengono gestiti con il protocollo SCSI (si ricordi quanto illustrato in sez. 5.1.1). Se la radice è su uno di questi dispositivi occorrerà inserire nel kernel il relativo supporto.

Multiple device driver support (RAID and LVM)
Questa sezione contiene le opzioni per abilitare il supporto del RAID software (vedi sez. 6.1) e del *Logical Volume Manager* (vedi sez. 6.2) ed in generale del sistema del *device mapper* che consente una serie di funzionalità avanzate (come l'uso di dischi cifrati). A meno di non avere la radice su uno di questi dispositivi il supporto può essere modulare. É presente in questa sezione anche il supporto per il *multipath* che consente di accedere tramite diversi nomi agli stessi dispositivi (ad uso dei dispositivi SAN).

IEEE 1394 (FireWire) support
Questa sezione contiene le opzioni per il supporto delle interfacce e dei protocolli per il bus IEEE 1394 (il cosiddetto *firewire*), un bus seriale veloce simile all'USB, usato sia per utilizzare dischi rimuovibili esterni che per la trasmissione dati da periferiche (viene usato principalmente per le telecamere).

Macintosh device drivers
Questa sezione contiene le opzioni per il supporto di alcune funzionalità specifiche ad uso delle macchine *Macintosh* prodotte dalla *Apple* (come l'emulazione dei tasti mancanti sul mouse).

Network device support
Questa sezione contiene le opzioni di configurazione per le varie schede di rete utilizzabili con Linux (sia Ethernet che di altro tipo), più il supporto per alcuni protocolli

di comunicazione di basso livello (PPP, SLIP, ecc.) e dispositivi virtuali. Il supporto può anche essere modulare, a meno di non avere la radice su un filesystem di rete. Questa è una delle sezioni più ampie ed è suddivisa in una lunga serie di sottosezioni relative ai vari driver disponibili.

ISDN support
Questa sezione contiene le opzioni per il supporto dei dispositivi ISDN e dei relativi protocolli.

Telephony support
Questa sezione contiene le opzioni relative al supporto di schede telefoniche dedicate, che consentono di telefonare direttamente dal computer, ed usare questo come ponte fra linea telefonica normale e VoIP.

Input device support
Questa sezione contiene le opzioni per il supporto per mouse, tastiere, joystick ed altri dispositivi analoghi.

Character devices
Questa sezione contiene le opzioni di configurazione per una lunga serie di dispositivi a caratteri. Qui è essenziale l'opzione per il supporto del terminale sullo schermo ("`Virtual terminal`") che serve per poter disporre all'avvio della *console* di sistema.[27] É sempre in questa sezione che si abilita il supporto per le porte seriali nella sottosezione "`Serial drivers`", e se si vuole la *console* sulla porta seriale si deve abilitare all'interno di questa l'opzione "`Console on 8250/16550 and compatible serial port`", ed il supporto dei terminali virtuali (quelli usati da tutti i programmi che emulano un terminale) con l'opzione "`Unix98 PTY support`".

Hardware Monitoring support
Questa sezione contiene le opzioni relative al supporto dei dispositivi di controllo di funzionamento dell'hardware presenti sulle schede madri moderne (ad esempio i sensori di temperatura).

Multimedia devices
Questa sezione contiene le opzioni di configurazione di *Video For Linux*, l'infrastruttura del kernel per i dispositivi video, che comprende il supporto per l'utilizzo di schede TV, schede di acquisizione video, webcam ed anche quello delle schede di sintonizzazione radio.

Graphics support
Questa sezione contiene le opzioni per il supporto di funzionalità relative alla grafica: l'uso del bus AGP ("`/dev/agpgart (AGP Support)`"), il supporto ("`Direct Rendering Manager`") per il DRI (*Direct Rendering Infrastructure*) di X11, l'interfaccia di accesso a basso livello alle funzionalità di accelerazione delle schede grafiche (vedi sez. 3.3.2), il supporto per i framebuffer e per la *console* sia su VGA che su framebuffer, come avviene per le macchine che non usano la VGA, come gli Apple, quello per i vari display montati sulle macchine e per il logo all'avvio.

[27] è il supporto per i terminali su *console*, che consente di usare i dispositivi `/dev/tty1`, `/dev/tty2`, ecc.

Sound card support
Questa sezione contiene le opzioni relative al supporto delle schede audio per Linux, sono presenti due sottosezioni, relative alle due modalità alternative di supporto disponibili, il vecchio OSS (*Open Sound System*), ormai deprecato ed in via di sostituzione il cui uso è da evitare, ed il nuovo ALSA (*Advanced Linux Sound Architecture*), È all'interno di dette sottosezioni che poi si potrà abilitare il supporto per le varie schede sonore.

HID Devices
Questa sezione contiene le opzioni per il supporto dell'interfaccia dei dispositivi di interazione "*uomo-macchina*", cioè tutto quello che attiene il cosiddetto HID (*Human Interface Device*), una astrazione che permette di usare in maniera generica dispositivi come il mouse.

USB support
Questa sezione contiene le opzioni relative al supporto dei vari chipset del bus USB (vedi sez. 5.4.4) e dei vari dispositivi che si possono inserire su di esso; è qui che si abilita il supporto per i dischi su USB che consente di usare le chiavette ed i dischi esterni.

MMC/SD/SDIO card support
Questa sezione contiene le opzioni relative al supporto delle varie carte *Secure Digital* e *MultiMediaCard* e dei chipset che consentono di leggerle.

Una volta completata la configurazione, qualunque sia il metodo con cui la si è effettuata, questa viene salvata su un nuovo `.config`. A questo punto si può passare alla compilazione del kernel e di tutti i moduli con il comando `make`. Si tenga presente però che questa è la procedura adottata a partire dal kernel 2.6, in precedenza erano necessari vari passi intermedi, che non tratteremo essendo questi kernel ormai questi completamente obsoleti.

La compilazione del kernel (e dei moduli) è in genere un processo piuttosto lungo e che utilizza pesantemente le risorse (memoria e CPU) della macchina. Pertanto viene spesso anche usato come test di carico. Il procedimento può essere reso più veloce usando l'opzione `-j` del comando `make` che consente di *parallelizzare* la compilazione.

Questa è una funzionalità del tutto generale di `make` (per la precisione della versione GNU) che consente di specificare come parametro dell'opzione un numero di processi da eseguire in parallelo, ciascuno dei quali compilerà parti indipendenti, così da avere in generale sempre qualche processo in compilazione anche quando gli altri sono bloccati sull'I/O, cosa che è particolarmente efficiente su macchine multiprocessore. Questo è possibile sfruttando appunto le informazioni sulle dipendenze usate da `make` per affidare a processi diversi la compilazioni di sorgenti indipendenti.

5.2.4 Installazione manuale del kernel e del *ramdisk* iniziale

Una volta eseguita con successo la compilazione i passi successivi riguardano l'installazione del nuovo kernel così ottenuto; il primo passo è quello di installare i moduli, questo viene fatto, usando un ulteriore *target*, con il comando `make modules_install`. I moduli vengono sempre installati sotto `/lib/modules`, in una directory diversa per ciascuna versione del kernel, con lo

stesso nome della versione del kernel (così come definita nel `Makefile`); questo significa che se si installano i moduli del kernel 2.6.26 i moduli e tutti i relativi file saranno installati in `/lib/modules/2.6.26/`.

I passi successivi sono l'installazione nel sistema della nuova immagine del kernel e dei file ad essa associati. Per questo vengono anche forniti altri *target* specifici per `make` (come `make install`, `make rpm` o `make deb-pkg`) che consentono di automatizzare l'installazione o generare dei pacchetti installabili, ma in generale è sempre possibile eseguire l'operazione manualmente, anche perché in questo caso le operazioni sono indipendenti dal *bootloader* (vedi sez. 5.3) che si intende usare.

Come spiegato in sez. 1.2.3 i file necessari all'avvio del sistema sono mantenuti in `/boot`, pertanto è qui che deve essere copiata l'immagine del kernel. Questa viene sempre creata nel file `vmlinux` nella directory dei sorgenti, ma per motivi storici nella architettura dei PC ordinari deve essere usata una immagine compressa che nel caso si trova, sia per le versioni a 32 che a 64 bit, sotto `arch/x86/boot/bzImage`. Su architetture hardware diverse si possono avere casi diversi a seconda del tipo di *bootloader* che viene utilizzato, ma in genere viene usato direttamente il file `vmlinux`.

La convezione adottata universalmente è quella di chiamare l'immagine del kernel posta sotto `/boot` con il nome `vmlinuz-`*`versione`*, dove la "*versione*" è il numero di versione impostato con le variabili del `Makefile` principale di cui si è parlato all'inizio di sez. 5.2.3, e che deve essere univoco qualora si voglia dare la possibilità al *bootloader* di utilizzare kernel diversi per l'avvio del sistema (tratteremo l'argomento in sez. 5.3.1).

Occorre inoltre copiare anche il file `System.map` che contiene le informazioni che permettono di identificare per nome, e non tramite un indirizzo in memoria, le varie routine interne del kernel. In questo file, la cosiddetta *mappa dei simboli*, viene mantenuto l'elenco completo dei nomi delle funzioni (i *simboli* appunto) che sono state "*esportate*", cioè rese visibili alle altre funzioni del kernel. La mappa contiene sia i simboli di quelle presenti nell'immagine di avvio che di tutte quelle disponibili all'interno dei moduli.

Il contenuto del file viene utilizzato dal kernel nella stampa degli eventuali errori riscontrati nell'esecuzione di una funzione interna. Le funzioni infatti sono accessibili al kernel solo con degli indirizzi binari, ma il file può essere utilizzato per ottenere i nomi degli indirizzi. Questa funzionalità di per sé non è essenziale, ma senza questa informazione diventa molto complesso per chi sviluppa il kernel capire dove si è verificato l'errore, ed anche più difficile per voi chiedere aiuto o cercare informazioni relativamente all'errore stesso. La sua assenza perciò non comporta problemi di funzionamento del sistema, ma solo una maggiore difficoltà per chi dovrà andare ad analizzare gli errori.

Lo stesso file è utilizzato anche in sede di installazione dei moduli quando vengono calcolate le dipendenze di un modulo da un altro, ed il comando `depmod` (che vedremo in sez. 5.2.5) darà degli errori in caso di sua mancanza o di non corrispondenza con il kernel attivo. Come per l'immagine del kernel anche `System.map` viene di norma copiato su `/boot` appendendo un "`-`" ed la versione; questo fa sì che ogni kernel sia in grado di trovare ed utilizzare la propria mappa dei simboli.

Infine è buona norma, tutte le volte che si installa un nuovo kernel, mantenere traccia anche delle opzioni di configurazione con cui lo si è prodotto, tanto che nella serie 2.6.x è possibile far generare direttamente al kernel stesso il contenuto di detto file, posto che si sia abilitato il relativo supporto durante la compilazione (opzione "`Kernel .config support`" in "`General setup`").

Target	Significato
`config`	interfaccia di configurazione a linea di comando.
`oldconfig`	interfaccia di configurazione a linea di comando, che riutilizza i valori precedentemente immessi.
`menuconfig`	interfaccia di configurazione a grafica testuale, basata sulle librerie `ncurses`.
`xconfig`	interfaccia di configurazione grafica.
`vmlinux`	crea solo l'immagine del kernel.
`modules`	compila soltanto i moduli.
`modules_install`	installa i moduli nella relativa directory.
`clean`	cancella tutti i file oggetto (i `.o`) presenti prodotti da una precedente compilazione.
`mrproper`	oltre a quanto esegue `clean`, cancella anche le informazioni sulle dipendenze.
`distclean`	oltre a quanto esegue `mrproper` e cancella ulteriori file prodotti cercando di riportare l'albero dei sorgenti identico allo stato immediatamente dopo la scompattazione.
`install`	esegue l'installazione del kernel.
`rpm`	crea un pacchetto RPM contenente l'immagine del kernel.
`deb-pkg`	crea un pacchetto Debian contenente immagine del kernel e moduli.

Tabella 5.19: Principali *target* del comando `make` per la compilazione del kernel.

Questo comporta la necessità di salvare il contenuto del corrispondente file `.config` e di norma lo si fa (questa ad esempio è la scelta di Debian) copiandolo sempre sotto `/boot`, ed assegnandogli, usando la stessa convenzione usata per la mappa dei simboli e l'immagine del kernel, un nome nella forma `config-`*`versione`*.

In tab. 5.19 si sono riportati i principali *target* di `make` usati per la compilazione e l'installazione del kernel dai sorgenti; in questo caso la documentazione non è disponibile in una pagina di manuale, ma si può trovare una lista completa dei vari *target* disponibili nel capitolo 10 di [LinuxNutshell].

Ricapitolando, l'insieme dei vari passi per ottenere un nuovo kernel ed installare tutti i file relativi, dovrà essere qualcosa del tipo:

```
make menuconfig
make
make modules_install
cp arch/x86/boot/bzImage /boot/vmlinuz-2.6.25.10-my
cp System.map /boot/System.map-2.6.25.10-my
cp .config /boot/config-2.6.25.10-my
```

dopo di che occorrerà configurare il proprio *bootloader* (vedi sez. 5.3) per l'uso del nuovo kernel.

Si tenga presente però che il procedimento appena illustrato può funzionare soltanto se nel kernel si è inserito direttamente il supporto necessario a montare la radice, vale a dire sia il supporto per la classe di dispositivi su cui si trova il filesystem (ed eventualmente per lo specifico hardware che ne consente l'accesso) che per il filesystem stesso. In caso contrario una volta lanciato il kernel non sarà in grado di montare la radice, e ci si troverà di fronte ad una brusca interruzione della procedura di avvio con un *kernel panic*.

Tutto questo, una volta che si conosca l'hardware specifico della propria macchina, non è un problema, a parte ovviamente la necessità di individuare quali sono le opzioni di configurazione che corrispondono al supporto per il proprio hardware, cosa talvolta non banale: basterà inserire all'interno del kernel il relativo supporto. Lo è però quando, come accade per i kernel standard forniti dalle distribuzioni, il kernel deve poter funzionare con una macchina generica. In tal caso infatti si dovrebbero inserire all'interno del kernel tutti i possibili driver per tutti i possibili tipi di hardware, ad esempio con le comuni macchine odierne sarebbe necessario il supporto per tutti i possibili chipset *Serial ATA*, la gran parte dei quali poi sarebbe inutilizzato ed occuperebbe inutilmente una risorsa preziosa come la RAM.

Benché sia possibile seguire questo approccio, e rimandare ad una successiva ricompilazione l'installazione di un kernel più adatto alla propria specifica macchina, la gran parte delle distribuzioni utilizza un approccio diverso che passa attraverso l'utilizzo di quello che in gergo viene chiamato un *ramdisk* iniziale, o *initrd*. In questo caso quello che si fa è inserire nel kernel soltanto il supporto per questo unico tipo di dispositivo che viene montato come radice provvisoria durante l'avvio.

È all'interno del *ramdisk* che viene installato un sistema minimale che si incarica di caricare i moduli necessari e compiere le operazioni necessarie ad accedere al dispositivo su cui si trova la radice. A questo punto sarà possibile montare la radice "*reale*" del sistema, e, tramite un apposito comando,[28] renderla la nuova radice, da cui eseguire l'ordinario avvio descritto in sez. 5.3.5, liberando al contempo la memoria utilizzata dal *ramdisk* e non più necessaria.

Perché questa procedura funzioni con un kernel generico occorre ovviamente poter generare un *ramdisk* appropriato. Nell'originario *initrd* questo veniva creato come immagine di un filesystem in cui si era inserito tutto il necessario. In genere venivano usati filesystem minimali come `cramfs` o `romfs` ed ovviamente doveva essere inserito direttamente nel kernel il relativo supporto. A partire dai kernel della serie 2.6.x invece è presente un nuovo meccanismo, chiamato *initramfs*, che fornisce il supporto per creare un *ramdisk* vuoto e popolarlo automaticamente con il contenuto di un archivio `cpio` compresso con `gzip`, non è pertanto necessario inserire nel kernel il supporto per nessun filesystem specifico, ma solo quello per l'uso del *ramdisk* iniziale. Questo è il meccanismo usato oggi da tutte le principali distribuzioni.

Qualunque sia il metodo occorre comunque poter creare un appropriato *ramdisk* iniziale che contenga, oltre a tutti i moduli che possono servire, anche tutti i programmi e le relative librerie che consentano di caricarli per accedere al dispositivo, montare la radice reale, e eseguire da questa la procedura di avvio ordinaria. Per questo motivo ogni distribuzione mette a disposizione un suo programma che consente di creare, dopo aver installato un nuovo kernel, un opportuno *ramdisk* ad esso associato.

Il comando usato originariamente sia su sistemi RedHat che Debian era `mkinitrd`.[29] Si trattava comunque, nonostante il nome fosse lo stesso, di programmi diversi funzionanti in maniera diversa. Tratteremo solo la versione di `mkinitrd` usata da RedHat e derivate dato che è la sola che sopravvive ancora oggi e che non è altro che uno script di shell che si cura della

[28] il comando si chiama `pivot_root`, come la omonima *system call*, ed è specifico di Linux; una volta che la radice "*reale*" sia stata montata in una directory all'interno di quella provvisoria il comando consente di rendere questa la nuova radice del sistema.

[29] si tenga presente che nonostante il nome, la versione in uso nelle distribuzioni recenti utilizza il nuovo meccanismo di *initramfs*, anche se quella usata da RedHat e CentOS è in grado di produrre *ramdisk* anche con il vecchio meccanismo per kernel della serie 2.4.x.

generazione del *ramdisk*. Il comando prevede due argomenti, il primo è il nome del file su cui verrà creato il *ramdisk* ed il secondo la versione del kernel per cui crearlo (per quella corrente è sufficiente usare il risultato di `uname -r`).

Il comando supporta una serie di opzioni che consentono di omettere alcune classi di moduli, per ridurre le dimensioni del *ramdisk* qualora si sappia per certo che queste non servano (ad esempio i moduli per i dischi SCSI). Si può inoltre specificare una lista di moduli da aggiungere o togliere, ed ulteriori opzioni per controllare il formato del *ramdisk* e le sue modalità di funzionamento, per le quali si può consultare la pagina di manuale. L'unica opzione che vale la pena citare è `-f` che consente di sovrascrivere un file esistente.

Nel caso di `mkinitrd` viene sempre generato un *ramdisk* che contiene i moduli strettamente necessari all'avvio della propria macchina, così come sono stati determinati durante l'installazione della stessa. Lo script di avvio del *ramdisk* si limita a caricali, in modo da montare la radice reale e poi far proseguire l'ordinaria procedura di avvio. Si deve pertanto stare attenti qualora si sposti il disco del sistema su un'altra macchina, perché non è detto che i moduli necessari siano gli stessi, nel qual caso non si sarebbe più in grado di avviare la macchina. Lo stesso dicasi qualora si volesse riutilizzare il kernel ed il relativo *ramdisk* su una macchina diversa.

Per superare questi problemi, e fornire un meccanismo che consenta l'avvio su una macchina generica, Debian e derivate hanno sostituito `mkinitrd` con `mkinitramfs`,[30] un programma che permette di creare un *ramdisk* in cui sono presenti tutti i moduli necessari ad effettuare l'avvio da un dispositivo generico. Questo significa che saranno presenti non solo i moduli per il supporto delle diverse schede SCSI o dei vari chipset per il *Serial ATA*, ma anche quelli per l'uso di dispositivi *virtuali* il RAID software (vedi sez. 6.1) o LVM (vedi sez. 6.2) e quelli necessari per l'avvio via rete. Inoltre saranno presenti anche i moduli per i vari filesystem utilizzabili per la radice, ed infine un sistema minimale con tutto l'occorrente per determinare quali sono i moduli da caricare per montare la radice reale ed eseguire l'avvio definitivo.

Per creare un nuovo *ramdisk* occorre invocare il programma specificando con l'opzione `-o` il file sul quale si vuole scriverlo. Se non si indica altro viene sottinteso l'uso del kernel corrente, altrimenti occorrerà indicarne esplicitamente la versione come argomento. Per le altre opzioni e la documentazione completa si consulti al solito la pagina di manuale.

Lo svantaggio di `mkinitramfs` è che il *ramdisk* iniziale creato è in genere di dimensioni maggiori, anche se è possibile indicare al programma di inserirvi soltanto lo stretto indispensabile o escludere alcune classi di moduli se si sa che questi non serviranno. Il vantaggio, oltre alla disponibilità dei vari moduli che consentono di avviare lo stesso kernel anche con macchine diverse, è quello che nel *ramdisk* è presente un sistema minimale dotato di una sua shell ed un sottoinsieme di comandi a cui si viene portati in caso di problemi, il che consente di operare manualmente e tentare un recupero se qualcosa non funziona.

Il comportamento di `mkinitramfs` è controllato dal rispettivo file di configurazione che è `/etc/initramfs-tools/initramfs.conf`, il formato del file è quello di una serie di dichiarazioni di variabili di shell, che controllano il comportamento di `mkinitramfs`, come è naturale dato che quest'ultimo è uno script di shell. In generale non c'è da modificare nulla, il file è comunque ben commentato ed anche per lui è disponibile una pagina di manuale.

Infine una ulteriore alternativa è costituita dal comando `yaird`, che a differenza dei precedenti nasce come progetto indipendente disponibile per qualunque distribuzione. La filosofia del

[30] nel caso di Debian è presente pure il programma `update-initramfs` che semplifica la creazione dei *ramdisk* per i kernel installati sotto `/boot`; entrambi con i loro file di configurazione fanno parte del pacchetto `initramfs-tools`.

programma è simile a quella di `mkinitrd`, cioè mira a creare un *ramdisk* iniziale di dimensioni minime che contenga soltanto lo stretto indispensabile. Contrariamente a `mkinitrd` il programma esegue una serie di controlli estesi, svolti tramite i dati ottenuti dal filesystem `/sys`, volti ad assicurarsi che il *ramdisk* prodotto sia adatto all'hardware effettivamente presente sulla propria macchina ed assicuri un avvio senza problemi, senza i quali si rifiuta di creare l'immagine.

Il comando deve essere invocato o con l'opzione `--output` tramite la quale si indica il file su cui produrre l'immagine, o con l'opzione `--test` che esegue soltanto un controllo sul sistema per verificare la capacità di produrre il *ramdisk*. Come argomento si deve specificare la versione di kernel per cui creare il *ramdisk* e come per `mkinitramfs` se non si specifica nessun argomento verrà usata la versione corrente. Per le altre opzioni ed i dettagli si rimanda di nuovo alla relativa pagina di manuale.

Qualunque sia il programma utilizzato, se si è deciso di optare per un kernel da avviare con l'ausilio di un *ramdisk*, l'ultimo passo dell'installazione manuale è sempre quello di copiare il file dell'immagine del *ramdisk* nella directory `/boot` con un nome adeguato (in genere si usa qualcosa del tipo `initrd-`*`versione`*) per poi essere in grado di farlo utilizzare al *bootloader* (vedi sez. 5.3.3 e 5.3.2).

5.2.5 La gestione dei moduli

Abbiamo già sottolineato come una delle caratteristiche più significative del kernel Linux sia il supporto modulare che consente di aggiungere funzionalità e supporto di periferiche anche dopo l'avvio caricando gli opportuni moduli. Ma oltre alla possibilità di effettuare questa operazione in maniera manuale, fin dalla nascita del supporto modulare era anche il kernel stesso ad essere in grado di effettuare autonomamente, ad esempio in corrispondenza della richiesta di una funzionalità mancante, il caricamento degli eventuali moduli ritenuti necessari. A partire dalla serie 2.6.x il meccanismo prevede che per il supporto ai dispositivi presenti i moduli siano caricati in corrispondenza al riconoscimento degli stessi.

Nelle prime versioni del kernel questo era realizzato attraverso un apposito demone, `kerneld`, che riceveva le richieste dal kernel ed eseguiva il caricamento dei moduli. A partire dalla serie 2.4.x il meccanismo è stato realizzato con un apposito sottosistema del kernel tramite il quale all'interno del kernel è possibile invocare una apposita funzione che dato il nome del modulo che si vuole caricare[31] si incarica di mettere in esecuzione un programma in user space in grado di eseguire tutte le operazioni necessarie al caricamento dello stesso.[32]

Per questo motivo, oltre a quanto illustrato nei paragrafi precedenti riguardo configurazione, compilazione ed installazione del kernel per il supporto modulare, per poter utilizzare a pieno quest'ultimo sono necessari anche tutti quei programmi in user space che permettono di gestire i moduli stessi (ed in particolare il loro caricamento) che sono raccolti in un apposito pacchetto che viene distribuito coi sorgenti del kernel.

Inoltre con i kernel della serie 2.6.x la gestione dei moduli è stata completamente ristrutturata, e a quanto appena illustrato è stata affiancata una ulteriore infrastruttura in user space (vedi

[31] la funzione si chiama `request_module`, ed in realtà con essa si potevano richiedere non soltanto dei moduli specifici ma anche delle funzionalità generiche, sempre identificate da un nome, che poi potevano essere fatte corrispondere anche a moduli diversi.

[32] il programma è `modprobe`, che tratteremo in dettaglio più avanti, ma si può utilizzare al suo posto un programma qualsiasi scrivendo il relativo pathname assoluto in `/proc/sys/kernel/modprobe`.

sez. 5.4.5) che oltre alla creazione dinamica dei file di dispositivo sotto la directory `/dev` sulla base dell'hardware effettivamente presente sulla propria macchina, fosse in grado di gestire la rilevazione dei dispositivi presenti, compresa quella occasionale dei dispositivi rimuovibili, ed il caricamento contestuale dei moduli necessari per il relativo funzionamento.

Il programma di base che consente di caricare un modulo all'interno del kernel, in modo da poterne usare le funzionalità, è `insmod`. Fino alla serie 2.4.x era il programma stesso che leggeva il codice dal disco, effettuava la risoluzione dei simboli, controllando che questi fossero compatibili con la versione del kernel in esecuzione, ed infine eseguiva il *collegamento* al codice del kernel. Con la serie 2.6.x il lavoro di controllo dei simboli ed il collegamento viene operato direttamente dal kernel stesso, per cui il comando non è che un programma minimale che legge il file contenente il codice del modulo che si vuole utilizzare e lo passa alla nuova versione della *system call* del kernel che gestisce il caricamento.[33]

Si tenga presente però che `insmod` consente di inserire nel kernel solo un modulo alla volta e che per farlo ha bisogno di risolvere tutti i simboli, che si ricordi sono le funzioni chiamate all'interno del codice del modulo, necessari al modulo stesso. Se alcuni di questi non sono presenti nel kernel in esecuzione, ma disponibili soltanto attraverso altri moduli non ancora caricati, il comando fallirà con un errore di "`unresolved symbol`". É per questo motivo che non si usa quasi mai `insmod` direttamente per caricare un modulo, ma lo si fa attraverso `modprobe`, che vedremo a breve.

Con i kernel della serie 2.6 il programma richiede semplicemente il pathname del file che deve essere caricato come modulo.Se un modulo li prevede, possono essere specificati come ulteriori argomenti di `insmod` dei parametri di configurazione nella forma `parametro=valore`. Si tenga presente che ogni argomento aggiuntivo viene interpretato come parametro, per cui si abbia cura di non inserire spazi prima e dopo l'assegnazione con "`=`".

Il nome del "*parametro*" dipende dal modulo stesso e la lista dei parametri disponibili per ciascun modulo si può ottenere tramite il comando `modinfo`, che tratteremo a breve, mentre il "*valore*" può essere a seconda dei casi una stringa, un numero intero, o una lista di numeri interi separati da virgole. I numeri inoltre sono specificabili sia in forma decimale scrivendoli normalmente (ad esempio `17`), che in forma ottale apponendovi uno zero (ad esempio `021` equivale di nuovo a `17`), che in forma esadecimale apponendovi uno `0x` (e ad esempio in questo caso si potrebbe scrivere il solito `17` come `0x11`).

Dato che `insmod` consente di caricare un modulo solo quando tutti i simboli di cui questo ha bisogno possono essere referenziati, se alcuni di essi sono presenti in un altro modulo non ancora caricato come accennato il comando fallirà. In casi come questo per poter andare avanti occorre identificare quale è questo modulo e caricarlo preventivamente, e se anche lui dipende da simboli presenti in altri moduli, ripetere la procedura risolvendo tutte le dipendenze. Dato che tutto questo può risultare assai scomodo, per automatizzare il caricamento dei moduli è stato creato un secondo comando, `modprobe`, che permette di risolvere le dipendenze in maniera automatica, trovare quali sono gli altri moduli che servono per poter utilizzare quello richiesto, e caricarli preventivamente.

Il meccanismo tramite cui `modprobe` è in grado di risolvere le dipendenze si basa sul contenuto del file `modules.dep` posto nella directory in cui sono installati i moduli. Questo file viene di

[33]questa funzione si chiama `init_module`, e si limita a prendere da un buffer il codice del modulo letto dal file, e gli eventuali parametri passati come stringa.

norma prodotto in fase di installazione degli stessi tramite il comando `depmod` (su cui torneremo più avanti) ed è costituito da una serie di righe con un formato del tipo:

```
modules.dep
/lib/modules/2.6.25/kernel/fs/ext2/ext2.ko: /lib/modules/2.6.25/kernel/fs/mbcache.ko
```

che ricalca esattamente quello del *target* di un `Makefile`; per ciascun modulo che ha delle dipendenze nel file è presente una riga contenente la lista degli altri moduli da cui esso dipende. In questo modo `modprobe` si incaricherà di caricarli preventivamente ed evitare il problema dei simboli non definiti.

Per trovare i file `modprobe` effettua la ricerca dei moduli da caricare fra quelli compilati per il kernel corrente, cercandoli nella directory `/lib/modules/‘uname -r‘`, dove vengono installati normalmente con l'esecuzione di `make modules_install` insieme a tutte le informazioni necessarie al loro utilizzo.

A partire dal kernel 2.4 tutti i moduli sono installati sotto la directory `kernel`, e all'interno di questa suddivisi per categorie in ulteriori directory, che sono:[34] `arch` per i moduli specifici per la propria architettura hardware, `drivers` per il supporto delle periferiche, `fs` per il supporto dei filesystem, `lib` per le funzioni di libreria, `mm` per la gestione della memoria, `net` per il supporto dei protocolli di rete, `crypto` per gli algoritmi di crittografia, `security` per le estensioni di sicurezza, `sound` per l'infrastruttura di gestione dell'audio. A loro volta i moduli installati sotto `drivers` sono suddivisi in altre sottodirectory per tipologia di hardware.

L'utilizzo più comune di `modprobe` prevede che gli si passi come argomento il nome del modulo da caricare, stavolta senza estensioni e senza fare riferimento ad un pathname; per comodità inoltre il comando non distingue fra i caratteri “`-`” e “`_`” qualora essi siano presenti nel nome del modulo. Se `modprobe` rileva la necessità di caricare altri moduli come dipendenze del modulo indicato provvederà a farlo in maniera automatica. Dato che il lavoro effettivo di caricamento del modulo è fatto dal kernel stesso, `modprobe` non è in grado di riportare nessun errore specifico sui motivi per i quali il caricamento fallisce; in questo caso sarà necessario consultare i messaggi del kernel nei file di log o con `dmesg`.

Come per `insmod` anche con `modprobe` si possono specificare negli argomenti successivi gli eventuali parametri da passare al modulo che viene caricato, il vantaggio di `modprobe` è che attraverso l'uso del suo file di configurazione, come vedremo a breve, si possono impostare dei valori di default da assegnare a questi argomenti, senza doverli più scrivere esplicitamente.

Oltre a caricare un modulo `modprobe`, se invocato con l'opzione `-r`, è anche in grado di rimuoverlo; in questo caso, se i moduli da cui quello rimosso dipende non sono utilizzati, il comando cercherà di rimuovere anche quelli. Inoltre a differenza dell'inserimento, se si richiede la rimozione si possono specificare negli argomenti anche più di un modulo alla volta. Si tenga presente però che se il modulo è utilizzato il comando fallirà.

Una seconda opzione importante è `-f` che consente di forzare l'inserimento di un modulo anche qualora questo non corrisponda alla versione del kernel corrente. Nel caso del kernel 2.6 il controllo avviene sia attraverso l'uso di apposite informazioni sulla versione salvate quando è stato abilitato il relativo supporto nel kernel (vedi sez. 5.2.3), che tramite una stringa di infor-

[34] come presenti al momento della stesura di questo testo, con riferimento al kernel 2.6.38.

mazioni (detta *vermagic*) che viene mantenuta comunque all'interno del modulo.[35] L'opzione consente di eliminare tutte queste informazioni, consentendo il caricamento del modulo. Dato che nel passaggio da una versione all'altra del kernel possono essere state introdotte delle differenze non banali una operazione di questo tipo è rischiosa, e può dar luogo, nella peggiore delle ipotesi, anche ad un crash del sistema.

Le altre opzioni principali del comando sono riportate in tab. 5.20, l'elenco completo ed una descrizione dettagliata delle stesse è come sempre disponibile nella pagina di manuale, accessibile con `man modprobe`.

Opzione	**Significato**
`-a`	consente di indicare come argomenti una lista di moduli da caricare.
`-C`	permette di usare un differente file di configurazione (da passare come parametro per l'opzione).
`-c`	stampa la configurazione di default.
`-f`	forza il caricamento del modulo anche qualora la versione non corrisponda a quella richiesta dal kernel corrente.
`-i`	ignora le direttive `remove` e `install` del file di configurazione.
`-n`	esegue tutta la procedura eccettuato il caricamento finale del modulo.
`-r`	rimuove il modulo specificato e l'insieme di moduli da cui esso dipende.
`-v`	abilita la stampa di un maggior numero di informazioni.

Tabella 5.20: Principali opzioni del comando `modprobe`

Come accennato il comportamento di `modprobe` è controllato da un opportuno file di configurazione. Con i kernel della serie 2.6.x la configurazione viene mantenuta in `/etc/modprobe.conf` ed alternativamente, se detto file non esiste, viene usato al suo posto il contenuto della directory `/etc/modprobe.d/`, considerando come configurazione la concatenazione di tutti i file in essa contenuti.

Sia `/etc/modprobe.conf` che i singoli file di `/etc/modprobe.d/` hanno un formato molto semplice in cui si specifica con una direttiva per riga nella forma di una parola chiave seguita da eventuali argomenti. Al solito righe vuote o che iniziano con il carattere "`#`" vengono ignorate. In tab. 5.21 si sono riportate le direttive disponibili.

La direttiva principale è `alias`, che permette di associare un nome (un *alias* appunto) ad un modulo, indicato con il nome del relativo file oggetto, ma senza estensione. In questo modo si può usare il nome dell'*alias* al posto di quello del modulo nella invocazione di `modprobe`. La direttiva svolgeva un ruolo fondamentale nella gestione dei moduli dei kernel della serie 2.4.x dato che in tal caso buona parte della gestione automatica dei dispositivi veniva realizzata richiedendo il supporto di una certa funzionalità attraverso l'uso del sistema di *kmod* che a sua volta richiedeva il caricamento dei moduli necessari invocando `modprobe` con un opportuno identificativo.[36]

Il problema è che questo identificativo non era associato direttamente al modulo necessario (anche quando questo era univoco) e quindi per ottenerne il caricamento occorreva impostare un

[35] sono disponibili anche due opzioni più specifiche, `--force-modversion` e `--force-vermagic`, per ignorare solo una delle due fonti di informazioni sulla versione.

[36] vale a dire che con `request_module` veniva eseguita una richiesta con un identificativo generico associato alla funzionalità voluta, a cui poi occorreva associare il modulo che la realizzava tramite una direttiva `alias`.

Direttiva	Significato
`alias`	definisce un nome da associare a un modulo, da passare rispettivamente come primo e secondo argomento; eventuali ulteriori argomenti saranno utilizzati come parametri per il modulo in fase di caricamento.
`blacklist`	richiede che il modulo passato come argomento venga ignorato dal sistema di caricamento automatico dei moduli.
`include`	consente di includere le configurazioni presenti nel file passato come argomento, questo può essere anche una directory, nel qual caso includerà le configurazioni definite nei file in essa contenuti.
`install`	consente a `modprobe` di eseguire il comando, passato come secondo argomento quando si richiede il caricamento del modulo indicato come primo argomento.
`options`	definisce, per il modulo passato come primo argomento, quali sono i parametri da utilizzare nel caricamento dello stesso passati negli argomenti successivi.
`remove`	analogo ad `install` ma il comando viene eseguito quando si richiede la rimozione di un modulo con `modprobe -r`.

Tabella 5.21: Le direttive del file di configurazione `modprobe.conf`.

alias. Inoltre c'era l'ulteriore problema di sapere quali erano gli identificativi utilizzati dal kernel per funzionalità e dispositivi, e a quali moduli farli corrispondere. Il kernel 2.6 ha reso tutto questo marginale in quanto tutte le volte che il modulo necessario per una certa funzionalità è univoco questo viene caricato direttamente, mentre buona parte del lavoro di identificazione dei dispositivi presenti e caricamento dei moduli necessari al loro uso è stato trasferito alla nuova infrastruttura per il rilevamento e la gestione dinamica dell'hardware che vedremo in sez. 5.4.5.

Per questo la direttiva `alias` viene usata per lo più qualora si voglia far caricare un modulo a `modprobe` usando un nome diverso. Resta possibile utilizzarla per associare uno specifico modulo ad una specifica richiesta effettuata dal kernel, solo che queste sono diventate molto meno comuni. Si tenga presente infine che non si possono definire degli `alias` a nomi definiti da un altro `alias`,mentre invece si possono specificare ulteriori argomenti nella direttiva, che verranno utilizzati come parametri da passare al modulo in fase di caricamento.

Una seconda direttiva molto utilizzata è `options`, che prende come argomenti il nome di un modulo seguito dai parametri da passare al suddetto quando questo viene caricato. Si tenga presente che i parametri così definiti verranno aggiunti a quelli eventualmente specificati con `modprobe` sulla riga di comando e a quelli eventualmente definiti in una direttiva `alias`. La direttiva è utile in quanto consente di impostare dei default per i parametri di un modulo in modo da non doverli scrivere tutte le volte che viene chiamato `modprobe`.

Una terza direttiva molto importante è `blacklist`, che nel caso in cui ci siano più moduli che supportano la stessa periferica, consente di evitare che uno di essi, indicato come argomento, venga utilizzato per il supporto della stessa nel caricamento automatico. Questa è l'unica direttiva che è specifica del kernel 2.6 a partire dal quale fra le informazioni presenti all'interno di ciascun modulo sono stati inseriti anche gli *alias* dei dispositivi e delle funzionalità che questo supporta (che possono essere visualizzati, come vedremo a breve, con il comando `modinfo`) così che il sistema di rilevamento e la gestione dinamica dell'hardware (vedi sez. 5.4.5) possa caricarli in maniera automatica.

Ci sono comunque dei casi in cui una stessa periferica è supportata da moduli diversi, ma si sa che uno di questi è preferibile agli altri, oppure casi in cui non si vuole che una certa periferica, pur essendo presente, venga attivata automaticamente. L'effetto della direttiva `blacklist` è

di far ignorare i valori degli *alias* interni al modulo passato come argomento, così che questo non sia mai caricato automaticamente, pur essendovi dell'hardware a cui esso è in grado di fornire supporto. Il contenuto tipico del file `/etc/modprobe.d/blacklist`, dove normalmente si inseriscono queste direttive, è infatti per lo più costituito da direttive riguardanti il supporto per le funzionalità di *watchdog* che non si vuole siano attivate se non viene esplicitamente avviato anche un demone che le gestisce. Si tenga presente che indipendentemente dalla presenza di una tale direttiva, sarà sempre possibile caricare il modulo con l'uso esplicito di `modprobe`.

Infine le due direttive `install` e `remove` consentono di sostituire rispettivamente il caricamento o la rimozione del modulo indicato come primo argomento con l'esecuzione del comando indicato come secondo argomento. In questo modo diventa possibile effettuare operazioni anche molto complesse (il comando può essere anche uno script di shell) contestualmente al caricamento o alla rimozione del modulo scelto. Occorrerà però, perché poi il modulo alla fine venga caricato o rimosso davvero, rieseguire `modprobe` all'interno del comando eseguito utilizzando l'opzione `-i` per disabilitare l'effetto di queste direttive,ovviamente solo se lo scopo del comando non è proprio quello di evitare il caricamento del modulo.

Come accennato per il funzionamento di `modprobe` occorre che nella directory dei moduli sia presente il file `modules.dep` che identifica le dipendenze fra i vari moduli. È grazie a questo file che è possibile determinare quali sono i moduli che contengono i simboli necessari ad un altro modulo, così da poter effettuare il caricamento di tutti quelli che servono nella giusta sequenza. Questo file viene creato tramite il comando `depmod` che esegue una scansione di tutti i moduli installati, identificando quali sono i simboli forniti ed utilizzati da ciascuno, per costruire così le relative dipendenze. Oltre a `modules.dep` il comando crea anche altri file, utilizzati dall'infrastruttura di *udev* (vedi sez. 5.4.5).

Se invocato senza argomenti `depmod` esamina i moduli relativi al kernel corrente, operando sotto `/lib/modules/`uname -r``, ma passandogli come argomento un numero di versione il comando utilizzerà i moduli relativi a tale versione, assumendo ovviamente che essi siano stati installati nella corrispondente directory sotto `/lib/modules`. Si possono passare come ulteriori argomenti una lista di file nel qual caso il comando si limiterà ad esaminare il contenuto degli stessi. In genere viene specificata anche l'opzione `-a`, che richiede il controllo di tutti i moduli, che comunque è sottintesa fintanto che non si specifica una lista di file.

In genere `depmod` viene invocato automaticamente quando si installano i moduli con `make modules_install` e non è necessario invocarlo di nuovo a meno che non si sia installato un qualche altro modulo in maniera indipendente, ad esempio se si sono installati in un secondo tempo dei moduli non presenti nel kernel che si sta utilizzando per ottenere qualche funzionalità aggiuntiva. Anche in tal caso comunque è molto probabile che nello script di installazione degli stessi sia prevista l'invocazione del comando.

L'unica opzione degna di nota è `-A` che prima di effettuare il calcolo delle dipendenze controlla preventivamente i tempi dei file, aggiornando `modules.dep` solo se qualcosa è cambiato. Per la descrizione dettagliata del comando, di tutte le altre opzioni e del file di configurazione `/etc/depmod.conf`, si può fare riferimento alle rispettive pagine di manuale accessibili con `man depmod` e `man depmod.conf`. Essendo il default del comando assolutamente appropriato all'uso ordinario, di norma `/etc/depmod.conf`, creato ad uso principalmente degli sviluppatori, non esiste.

Uno degli aspetti rilevanti della gestione dei moduli come accennato è la possibilità di controllarne alcune caratteristiche di funzionamento tramite il passaggio di opportuni parametri

in fase di caricamento. In questo caso il problema maggiore è identificare detti parametri ed impostarne un valore appropriato. Ma se riuscire a determinare il significato dei parametri e dei rispettivi valori può richiedere ricerche più o meno complesse nella documentazione del kernel, se non addirittura la consultazione diretta del codice, per poter iniziare a cercare occorre quantomeno sapere quali sono i parametri supportati.

A questo scopo diventa molto utile il comando `modinfo` che consente di esaminare il file oggetto del modulo passato come argomento ed estrarne una serie di informazioni, la principale delle quali è la lista dei parametri supportati dal modulo. Un esempio del comando è il seguente:

```
anarres:~# modinfo 8139cp
filename:       /lib/modules/2.6.26-1-686/kernel/drivers/net/8139cp.ko
license:        GPL
version:        1.3
description:    RealTek RTL-8139C+ series 10/100 PCI Ethernet driver
author:         Jeff Garzik <jgarzik@pobox.com>
srcversion:     5CF4FE9D06468F0B10B2660
alias:          pci:v00000357d0000000Asv*sd*bc*sc*i*
alias:          pci:v000010ECd00008139sv*sd*bc*sc*i*
depends:        mii
vermagic:       2.6.26-1-686 SMP mod_unload modversions 686
parm:           debug:8139cp: bitmapped message enable number (int)
parm:           multicast_filter_limit:8139cp: maximum number of filtered multicast addresses (int)
```

L'output del comando riporta una serie di campi, identificati per nome e seguiti da un valore. Nel caso dei parametri del modulo i campi di interesse sono quelli di tipo `parm`; nell'esempio allora vediamo come il modulo `8139cp` abbia due parametri con valore intero, `debug` e `multicast_filter_limit`. Si noti anche come il comando riporti anche le dipendenze del modulo (nel campo `depends`) ed una serie di campi `alias` che dicono a quali classi di dispositivi o funzionalità il modulo è in grado di fornire supporto.[37]

Se invocato con l'opzione `-k`, seguita da un identificativo di versione, il comando esaminerà il modulo disponibile per quella versione di kernel, piuttosto che quello per il kernel corrente; la cosa può risultare utile se si vogliono esaminare le informazioni di un kernel che si è installato ma non si sta usando. Oltre a questa il comando supporta una serie di opzioni che gli permettono di stampare solo alcune delle informazioni disponibili, al solito per la descrizione completa si può fare riferimento alla pagina di manuale.

Una volta che un modulo non sia più utilizzato, se ne può richiedere la rimozione. Abbiamo visto a come questo possa essere fatto in maniera "*intelligente*" con l'opzione `-r` del comando `modprobe`, che rimuove anche le eventuali dipendenze. Questa funzionalità però è presente in questa forma solo a partire dal kernel 2.6, in precedenza era previsto un comando apposito, `rmmod`, che resta ancora come comando di basso livello per la rimozione dei moduli, nello stesso senso in cui lo è `insmod` per l'inserimento.

Il comando prende come argomento il nome di un singolo modulo di cui esegue la rimozione. Ovviamente perché questa avvenga con successo il modulo in questione non deve essere in uso, né contenere simboli usati da un altro modulo (cioè non devono esserci altri moduli che *dipendano* da esso).

[37] il nome è dovuto al fatto che questi campi fanno riferimento proprio ai nomi usati nella direttiva `alias` di `modprobe.conf` per associare un modulo ad una certa funzionalità o dispositivo, e vengono usati dal sistema di *udev* (vedi sez. 5.4.5) per il caricamento automatico dei moduli.

È comunque possibile, a proprio rischio e pericolo, forzare la rimozione con l'opzione `-f`, ma per poterla usare si deve avere abilitato in fase di compilazione del kernel l'opzione `Forced module unloading`; una rimozione forzata resta comunque una operazione estremamente pericolosa, perché rimuovendo un modulo in uso il relativo codice sparisce dal kernel nonostante vi siano riferimenti ad esso, con effetti che possono molto facilmente causare il crollo completo del sistema.

Alternativamente si può usare l'opzione `-w` che blocca ogni ulteriore utilizzo del modulo indicato, ma prima di effettuarne la rimozione si ferma in attesa che questo venga liberato da ogni uso residuo. Per l'elenco completo delle opzioni si consulti al solito la pagina di manuale.

Infine per consultare la lista dei moduli caricati in memoria, e ottenere le informazioni relative al loro stato si può utilizzare il comando `lsmod`. Il comando non prende opzioni né argomenti e con il kernel 2.6 non è altro che un semplice programma che formatta in maniera più leggibile il contenuto del file `/proc/modules`. Un esempio del risultato di questo comando è il seguente:

```
anarres:~# lsmod
Module                  Size  Used by
cp2101                 11428  0
usbserial              26472  1 cp2101
i915                   25280  2
drm                    65256  3 i915
binfmt_misc             7528  1
tun                     8292  0
ppdev                   6468  0
parport_pc             22500  0
lp                      8164  0
parport                30988  3 ppdev,parport_pc,lp
autofs4                16420  0
...
```

La prima colonna indica il nome del modulo, mentre la seconda le sue dimensioni in memoria. La terza colonna infine indica se il modulo è in uso ed è divisa in due parti; all'inizio viene riportato il numero di riferimenti presenti, che dice quante volte il modulo è utilizzato, a questo segue, se ve ne sono, la lista dei nomi degli altri moduli che dipendono da esso. Un valore nullo del numero di riferimenti indica che il modulo non è utilizzato e può essere rimosso.

Il comando consente di verificare se un modulo è in uso, operazione che è utile fare prima di richiedere una rimozione che altrimenti fallirebbe. Inoltre con `lsmod` si può anche verificare se un modulo è effettivamente utilizzato, o risulta occupato soltanto a causa della presenza di dipendenze da esso da parte di altri moduli. In quest'ultimo caso, avendone la lista, li si potranno rimuovere preventivamente liberando il modulo originario, che a questo punto potrà essere rimosso a sua volta.

Infine un altro file utilizzato da varie distribuzioni nella gestione dei moduli è `/etc/modules`, in cui si può inserire la lista dei moduli che si vuole siano caricati all'avvio del sistema. Il formato del file è estremamente semplice, ogni linea deve contenere il nome di un modulo, al solito le linee vuote o che iniziano per "`#`" vengono ignorate. Un possibile esempio del contenuto di questo file, preso dalla versione installata su una Debian Etch, è il seguente:

/etc/modules

```
# /etc/modules: kernel modules to load at boot time.
#
# This file should contain the names of kernel modules that are
```

```
# to be loaded at boot time, one per line.  Comments begin with
# a #, and everything on the line after them are ignored.
#ide-floppy
auto
#
# I2C adapter drivers
i2c-isa
i2c-ali15x3
# I2C chip drivers
w83781d
eeprom
```

In genere si utilizza questo file per forzare il caricamento di un modulo all'avvio senza aspettare che questo venga caricato automaticamente.

5.3 La gestione dell'avvio del sistema

In questa sezione prenderemo in esame la procedura di avvio del sistema, dettagliando i vari passaggi che portano dall'accensione della macchina al pieno funzionamento del sistema, e come si possa intervenire (trattando i vari programmi coinvolti ed i relativi meccanismi di configurazione) a ciascuno stadio di questa procedura.

5.3.1 L'architettura dell'avvio del sistema

Una delle caratteristiche comuni di tutti i computer moderni è che questi devono prevedere una apposita procedura di avvio che porti dalla macchina spenta ad un sistema attivo o funzionante. Questo consiste in quel processo che in inglese viene chiamato *bootstrap*, usando un'espressione gergale che fa riferimento al paradosso del "*sollevarsi da terra tirandosi per i lacci degli stivali*", che sembra analogo al fatto di riuscire a far eseguire un programma ad un computer che quando parte non ne contiene alcuno.

In realtà questo non è vero ed in tutte le piattaforme hardware è prevista la presenza di un programma iniziale, che serve appunto a gestire il *bootstrap* della macchina. Il successo della procedura è dovuto ad una caratteristica comune di tutti i processori, che prevede che quando questi vengono accesi inizino ad eseguire il programma posto ad una specifica locazione di memoria. È cura di chi assembla la macchina far corrispondere questa posizione di memoria al suddetto programma di gestione dell'avvio, che viene mantenuto nella memoria non volatile di cui ogni computer moderno è dotato, la ROM, o più precisamente la EPROM (in quanto riscrivibile). In genere si fa riferimento a questo programma con il nome di *firmware*, come via di mezzo fra *hardware* e *software*.

In generale ogni piattaforma hardware ha un suo *firmware* specifico per quella stessa piattaforma, ed esiste anche una versione libera e multi-piattaforma di *firmware*, *openfirmware*, creato dalla Sun ed usato anche da Apple e IBM per piattaforme diverse da quelle degli ordinari PC compatibili. Nei comuni PC il *firmware* è più noto con il nome di BIOS, acronimo di *Basic Input/Output System*, che è il nome dato al *firmware* nei primi PC prodotti dalla IBM.

I *firmware* comunque sono programmi, per cui è sempre possibile avere delle versioni alternative, che, viste le crescenti capacità dell'hardware moderno, possono essere molto più sofisticati

del BIOS tradizionale. Ad esempio per alcune piastre madri è disponibile un *firmware* alternativo, *coreboot*, nato per far eseguire direttamente il kernel invece del BIOS e velocizzare la procedura di avvio, poi trasformatosi in un sostituto del BIOS in grado di lanciare altri *firmware* o *bootloader* oltre che Linux stesso.

Oggi la quasi totalità dei PC nuovi viene fornita con UEFI (*Unified Extensible Firmware Interface*), un altro *firmware* alternativo rispetto al BIOS, le cui specifiche sono state create da inizialmente da Intel, ma che oggi viene gestito dallo *UEFI forum* che vede fra i promotori i principali produttori di processori e di sistemi operativi proprietari. Un *firmware* UEFI fornisce a sua volta una interfaccia standardizzata che permette di gestire tutta la procedura di avvio in maniera molto più flessibile rispetto al vecchio BIOS. Tutti i *firmware* che supportano UEFI prevedono comunque anche la possibilità di partire nella cosiddetta modalità classica, che utilizza le stesse modalità con cui un BIOS tradizionale permette di lanciare un sistema operativo, per cui tratteremo per prima queste.

Benché con il BIOS le modalità di avvio dipendano dalla versione disponibile (in pratica ogni produttore ha realizzato una sua variante), una serie di caratteristiche sono comuni, e tutti i BIOS più recenti permettono di avviare il sistema da una varietà di dispositivi, come floppy (se ne trovate ancora di funzionanti), disco e CDROM. Altre modalità di avvio supportate da tutte le macchine recenti, ma che potrebbero essere assenti su quelle molto datate, sono via rete o da chiavetta USB.

In genere il dispositivo da cui effettuare l'avvio si può selezionare tramite l'apposito menu di configurazione del BIOS che permette di determinare la sequenza in cui vengono controllati tutti i dispositivi da cui questo è possibile. Il primo su cui si trovano dati opportuni verrà utilizzato, per cui in genere si mettono in cima alla lista i dispositivi rimuovibili, così da non dover cambiare la configurazione e poterli utilizzare semplicemente inserendoli nella macchina.

La procedura con cui il BIOS può caricare in memoria il sistema operativo dipende dal supporto da cui quest'ultimo viene prelevato, nel caso di un floppy il meccanismo è molto semplice: il BIOS legge il primo settore ed esegue il programma ivi contenuto, che è quello che deve incaricarsi di eseguire l'avvio. Per i CDROM il meccanismo è lo stesso, solo che detto settore deve essere letto dalla immagine di un floppy presente all'interno del CD, in sostanza il BIOS vede solo l'immagine del floppy contenuta nel CD, che tratta come tale, che è disponibile qualora questo sia stato masterizzato con le opportune estensioni (le *El Torito*, vedi sez. 5.1.7).

Il caso più comune comunque è quello in cui l'avvio viene fatto dal disco rigido, un tempo la cosa poteva dipendere dal tipo di disco, se IDE (ATA o *Serial ATA*) o SCSI e per questi ultimi dal supporto del relativo controller, oggi tutti i BIOS trattano indifferentemente i vari tipi di disco, comprese le chiavette USB che vengono viste come normali dischi, secondo una modalità denominata *USBHDD*.[38] Qualora i dischi disponibili siano più di uno è cura del BIOS fornire un menù di configurazione per impostare quale usare.

In questo caso la procedura prevede che il BIOS legga il primo settore del disco scelto per l'avvio, che anche per questo motivo viene chiamato *Master Boot Record*, ma a differenza del caso del floppy/CDROM, in cui il primo settore contiene solo il codice da eseguire, nell'MBR è presente anche la tabella delle partizioni, ed il codice posto in esecuzione deve essere stato installato in una posizione opportuna.

[38] esiste una modalità alternativa denominata *USBZIP*, usata per avviare da dischi estraibili ZIP su USB, ormai dismessa ma che su alcune vecchie macchine può essere l'unica disponibile.

Infine nel caso sia disponibile l'avvio via rete questo viene eseguito tramite il supporto PXE (*Pre-boot eXecution Environment*) che consente di ottenere l'immagine del *bootloader* (che in questo caso viene chiamato NBP, *Network Bootstrap Program*) e del kernel da mettere in esecuzione da un server di rete tramite gli opportuni protocolli. In questo caso occorre aver installato su un'altra macchina i servizi TFTP e DHCP (tratteremo quest'ultimo in sez. 8.2.3) per mettere a disposizione tutti i dati necessari.

Indipendentemente dal supporto scelto, la caratteristica generale resta quella che il BIOS non è in grado di lanciare direttamente il sistema operativo e deve ricorrere ad un programma esterno, chiamato *bootloader*, il quale deve essere stato opportunamente installato sui dispositivi utilizzati per l'avvio. Questa è una limitazione specifica del BIOS che da sempre ha previsto l'uso di un *bootloader* separato per effettuare l'avvio del sistema operativo, limitazione che non è presente nei *firmware* di altre piattaforme hardware che sono in grado di lanciare direttamente il sistema operativo, in genere caricandolo da una apposita partizione, con una gestione che per questo risulta notevolmente semplificata.

In generale il *bootloader* è un programma elementare il cui unico compito è quello di trovare il sistema operativo, caricarlo in memoria ed eseguirlo, passandogli eventuali parametri di avvio. Nel caso di Linux questo significa trovare l'immagine del kernel e, se lo si usa, del *ramdisk*. Si tenga presente che un *bootloader* oltre che nel *Master Boot Record*, può essere installato anche nel settore iniziale di una partizione. Questo permette allora di concatenare più *bootloader* in modo che il primo lanci il successivo, e così via, ed in particolare questa è la tecnica usata normalmente per lanciare il *bootloader* di Windows da parte dei *bootloader* usati per Linux.

L'introduzione di UEFI come sostituto del BIOS è servita principalmente a superare l'ostacolo costituito dal limite di dover mantenere il codice di partenza del *bootloader* nel limitato spazio disponibile nel *Master Boot Record*, ed al contempo fornire una piattaforma di avvio avanzata e generica che consentisse di supportare una serie di funzionalità aggiuntive, come le partizioni GPT (viste in sez. 5.1.2), il cosiddetto *secureboot*, ecc.

Per superare i limiti dell'uso del *Master Boot Record* è stata utilizzata la più classica delle soluzioni: aggiungere un altro livello di redirezione. A differenza del BIOS classico infatti un *firmware* UEFI non esegue nessun codice dall'MBR, e benché in modalità di compatibilità la maggior parte possano eseguire un avvio classico, l'avvio in modalità nativa richiede la presenza di una partizione riservata, denominata ESP ("*EFI System Partition*") in cui installare delle opportune "*EFI Application*"; sono queste che vengono eseguite dal *firmware* e che possono effettuare tutte le successive operazioni, come caricare un *bootloader* ed avviare il sistema.

Una "*EFI System Partition*" deve contenere un filesystem di tipo FAT (ufficialmente è richiesto FAT32, anche se nella gran parte dei casi va bene anche FAT16) ed essere marcata come tale. Anche se è possibile usare una partizione classica, che nel caso deve essere di tipo `EF`, è comunque opportuno (obbligatorio nel caso si debba avere un dual boot con Windows) usare le nuove partizioni GPT che sono state introdotte proprio con le specifiche di EFI. Una ESP in tal caso viene identificata dal GUID `C12A7328-F81F-11D2-BA4B-00A0C93EC93B`, e viene impostata da `gdisk` con il tipo `EF00` (si ricordi quanto detto in sez. 5.1.2).

All'avvio un *firmware* UEFI esegue le inizializzazioni, recupera i dati che gli dicono quale applicazione lanciare per l'avvio (se li si sono impostati dall'interfaccia di gestione interna) e poi la esegue. In assenza dell'impostazione di una applicazione specifica (cosa che non tutti i *firmware* consentono) viene eseguita una scansione dei dispositivi presenti (come CD, dischi, chiavette) cercando all'interno degli stessi eventuali partizioni ESP, per eseguire il programma di

avvio di default che è sempre `/efi/boot/boot`*`arch`*`.efi` dove *`arch`* indica l'architettura hardware ed in genere è una sigla breve come `ia32`, `x64`, `arm` (rispettivamente per le più comuni piattaforme come PC a 32 bit, a 64 bit e ARM).

L'utilizzo di una partizione dedicata consente di installare ulteriori applicazioni in grado di fornire le funzionalità più varie. Se poi il proprio *firmware* UEFI fornisce una interfaccia nativa per farlo si potrà scegliere quale di queste applicazioni lanciare direttamente all'avvio, oppure si potrà installare direttamente (col nome di default citato prima) un *boot manager* come *rEFInd*, che consente di scegliere lui quale sistema operativo o altra applicazione EFI lanciare sulla base di quanto installato. Inoltre è anche possibile installare direttamente delle applicazioni specifiche come una "*UEFI shell*" per effettuare operazioni amministrative prima di lanciare il sistema operativo.

In questo modo ogni sistema operativo può installare un suo *bootloader*, ed una convenzione vuole che ciascuno utilizzi una sua sotto-directory specifica; ad esempio Windows usa `/efi/Microsoft`, mentre le distribuzioni Linux usano `/efi/NomeDistribuzione/`. Nel caso di Linux l'applicazione EFI inizialmente utilizzata per l'avvio era ELILO, ma oggi è disponibile, ed usata praticamente da tutte le distribuzioni, una versione EFI di GRUB (vedi sez. 5.3.3) che in questo caso sarà lanciato come applicazione EFI invece che direttamente dal BIOS.

Pertanto, sia che si usi un BIOS tradizionale, o che si usi UEFI, nell'architettura dei PC l'avvio del sistema prevede comunque dopo l'esecuzione del *firmware* un secondo stadio in cui si passa attraverso l'uso di un *bootloader* il cui compito è quello di caricare il kernel in memoria e di eseguirlo. Trattandosi di nuovo di un programma, anche in questo caso esistono diverse alternative (tratteremo nel seguito le più rilevanti), ma il compito da eseguire è sostanzialmente lo stesso, fornire una interfaccia in cui si possa scegliere il kernel che si vuole utilizzare (in genere Linux, ma potrebbe essere anche un kernel BSD, o un programma diagnostico come `memtest`) e metterlo in esecuzione.

Oltre a questo i *bootloader* consentono di passare eventuali argomenti al kernel che mettono in esecuzione e di caricare l'immagine di un *ramdisk* per fornire un filesystem iniziale con gli strumenti minimi necessari per individuare e montare il supporto su cui è stato effettivamente installato tutto il sistema operativo (torneremo su questo in sez. 5.3.4). Solo a questo punto si passerà allo stadio successivo, in cui il kernel viene effettivamente eseguito.

Una volta messo in esecuzione Linux effettuerà una serie di operazioni preliminari, come le inizializzazioni delle infrastrutture generiche e dei dispositivi il cui supporto è stato compilato direttamente nel kernel. Una volta completata l'inizializzazione dell'hardware tutto quello che resta da fare è semplicemente montare il dispositivo su cui si trova la directory radice e lanciare il programma di avvio del sistema, che di norma è `/sbin/init`.

Nel caso si stia utilizzando un *ramdisk* di avvio, verrà montato questo come directory radice e lanciato lo `/sbin/init` del *ramdisk*, con un ulteriore stadio prima del lancio del programma di avvio finale. Le operazioni comunque, per quanto riguarda il kernel, si concludono con il lancio della prima istanza del programma di avvio, tutto il resto verrà eseguito in *user space*.

5.3.2 Il *bootloader* SYSLINUX

Il *bootloader* SYSLINUX viene per lo più utilizzato per creare le immagini di avvio usate per floppy, CDROM e chiavette USB. Per i CD è disponibile anche la variante ISOLINUX, che supporta la stessa sintassi del file di configurazione ma non necessita di una immagine di un

filesystem FAT e può usare direttamente il contenuto del CD. In genere è il *bootloader* usato nei dischi di installazione di molte distribuzioni (ad esempio Debian). Per questo scopo viene utilizzato anche sulle chiavette USB. Inoltre la variante PXELINUX è molto diffusa come programma per gestire l'avvio via rete. Tutte queste varianti condividono la stessa sintassi generale del file di configurazione, e la stessa forma e funzionalità del prompt di avvio, per questo buona parte di quanto illustrato in questa sezione può applicarsi anche alle altre varianti.

Il programma deve la sua popolarità alla relativa semplicità con cui consente di creare immagini di avvio, in una tipologia di utilizzo cioè in cui non è molto importante avere una grande flessibilità nel cambiare la versione di kernel da avviare, come avviene appunto nel caso di un CDROM o di una chiavetta di installazione.

Uno dei suoi punti di forza è che i dati da cui creare l'immagine d'avvio vengono mantenuti in un filesystem FAT, cosa che lo rende facilmente utilizzabile anche per l'uso diretto su chiavette USB che in genere vengono formattate in questo modo, che possono essere rese avviabili semplicemente copiandoci i vari file. L'uso di un filesystem FAT rende inoltre molto semplice modificare una precedente immagine di avvio, in quanto basterà montarla (direttamente o in *loopback* se la si è messa su un file) per accedere ai relativi contenuti.

Per creare una immagine d'avvio basterà installare i file necessari sul filesystem FAT con il comando `syslinux`, e nel caso di un floppy questo è quanto basta. Nel caso di una chiavetta USB (o in generale di un disco) invece sarà necessario sovrascrivere il *Master Boot Record* con la versione fornita dal insieme al programma (il file è `mbr.bin`, che nel caso di Debian viene installato sotto `/usr/lib/SYSLINUX`). In tal caso viene richiesta la sovrascrittura con l'uso di `dd`, in sostanza per installare il tutto sul disco `/dev/sdx` si dovranno usare i comandi:

```
# dd conv=notrunc bs=440 count=1 if=/usr/lib/SYSLINUX/mbr.bin of=/dev/sdX
# syslinux /dev/sdX1
```

L'unica opzione significativa è `-s` che installa una versione "*safe, slow and stupid*" che funziona anche con alcuni BIOS difettosi. Se ci si limita a seguire i passi precedenti all'avvio SYSLINUX caricherà l'immagine del kernel dal file `linux`, che si suppone sia stato copiato a sua volta sul disco (ovviamente per farlo occorrerà semplicemente montarlo ed effettuare la copia); questo comportamento può essere personalizzato usando il file `syslinux.cfg`, che deve essere anch'esso copiato nella radice del filesystem FAT (è comunque supportata anche la lettura dai file `/boot/syslinux/syslinux.cfg` e `/syslinux/syslinux.cfg`).

Il formato del file, che è identico anche per le varianti ISOLINUX e PXELINUX, segue la convenzione di ignorare righe vuote e quanto segue un "`#`", mentre le configurazioni devono essere indicate da righe inizianti con una parola chiave seguita da uno o più parametri posti sulla stessa riga separato da spazi.

La direttiva `label` seguita da un nome consente di etichettare diverse immagini del kernel da avviare, queste devono essere specificate con la direttiva `kernel` seguita dal pathname del file. Si tenga presente che per i nomi delle etichette valgono alcune limitazioni presenti per i nomi dei file MSDOS, ad esempio usare etichette con più di un "`.`" nel nome farà ignorare quanto segue il secondo punto. La direttiva `initrd` consente di specificare un *ramdisk*; eventuali argomenti del kernel devono essere specificati con una direttiva `append`. Un riassunto delle principali direttive è riportato in tab. 5.22. Per i dettagli si può fare riferimento alla pagina di manuale di `syslinux` al solito accessibile con `man syslinux`.

Direttiva	Significato
`append`	imposta l'elenco di argomenti da passare al kernel, eventuali altri argomenti inseriti al prompt verranno accodati a questi (consentendo una sovrascrittura), se specificata prima della prima `label` si applica globalmente.
`default`	imposta il kernel da usare di default, indicato tramite l'etichetta (fra quelle specificate in una successiva `label`) ad esso associata, se non specificato viene usato il valore `linux`.
`display`	pathname di un file di testo il cui contenuto viene stampato sulla schermata del prompt di avvio.
`f1,...,f12`	pathname di un file il cui contenuto viene stampato al prompt di avvio in conseguenza della pressione dei tasti F1, ..., FN.
`initrd`	imposta il pathname all'immagine del *ramdisk* che verrà caricato all'avvio.
`kernel`	imposta il pathname all'immagine del kernel che verrà caricata all'avvio.
`label`	imposta l'etichetta che identifica una immagine di kernel, richiede che si specifichi di seguito una direttiva `kernel` per indicare l'immagine del kernel ed ulteriori direttive che verranno usate qualora si usi quell'etichetta come valore per `default` o la si scriva sul prompt di avvio.
`prompt`	il valore 1 richiede di mostrare sempre all'avvio il prompt della riga di comando *syslinux*, in caso di valore nullo o in assenza della direttiva il prompt viene mostrato solo premendo uno dei tasti di controllo (Alt, Shift, ecc.).
`timeout`	imposta il tempo di attesa (in decimi di secondo) per accedere al prompt di avvio che parta il caricamento del kernel; un valore nullo (il default) inizia immediatamente il caricamento.

Tabella 5.22: Direttive di configurazione di `syslinux.cfg`.

All'avvio del sistema SYSLINUX mette a disposizione una riga di comando (esplicitamente o con la pressione di uno dei tasti di controllo (Alt, Shift, ecc.) con un prompt da cui è possibile scegliere quale immagine usare fra quelle disponibili. Questa può essere scelta indicandola con il valore usato nella corrispondente direttiva `label` della configurazione, altrimenti si può usare il valore di default linux, ed eventualmente aggiungere ulteriori parametri di avvio del kernel, che andranno ad aggiungersi a quelli eventualmente impostati nella configurazione di `syslinux.cfg`.

5.3.3 I *bootloader* GRUB e GRUB2

Uno dei più sofisticati *bootloader* disponibili per Linux, è GRUB, nome che sta per *Grand Unified Bootloader*. GRUB è nato come *bootloader* per il sistema HURD[39] ma è in grado di avviare qualunque altro tipo di sistema (compreso Linux e BSD). Benché sia ancora usato sta venendo progressivamente sostituito dalla sua evoluzione, GRUB2.

La caratteristica principale di GRUB è che è formato da diverse parti, dette *stadi*, ciascuno dei quali, come gli omonimi dei razzi, viene usato per lanciare il successivo. In sostanza poi gli stadi sono due, il primo viene installato sull'MBR ed il suo unico compito è quello di lanciare lo stadio successivo, che poi si incaricherà di effettuare tutte le operazioni successive. Il grande vantaggio di GRUB è che siccome il primo stadio deve solo occuparsi di trovare il secondo ed è quest'ultimo che fa tutto il lavoro, tutte le restrizioni che si applicano ad un programma che deve stare nell'MBR scompaiono.

[39] HURD è un sistema libero sperimentale realizzato dalla FSF, basato su una architettura microkernel, con funzionalità teoricamente molto avanzate; ma il cui sviluppo è ancora molto limitato.

Per questo GRUB è in grado di leggere nativamente i principali filesystem, cosicché non solo si può fare riferimento alle immagini del kernel attraverso un pathname, ma si ha anche a disposizione una micro-shell che consente di navigare attraverso un filesystem ed esplorarne il contenuto. Questo vuol dire che basta cambiare il file di configurazione di GRUB o sovrascrivere una immagine del kernel perché la nuova versione sia usata al successivo riavvio, e non è necessario eseguire la reinstallazione nel *Master Boot Record* come per *bootloader* primitivi come era LILO, evitando i guai che nascono tutte le volte che ci si dimenticava di farlo. Un altro grande vantaggio è che GRUB può ricevere i comandi all'avvio non solo dalla *console*, ma anche via seriale e via rete, dato che non ha bisogno del BIOS per gestire l'I/O, il che lo rende molto più flessibile.

Tutti i file di GRUB sono mantenuti in `/boot/grub`, in questa directory si trovano i file `stage1` e `stage2` che contengono i due stadi standard usati nell'avvio, e una serie di file `*_stage1_5` che contengono gli stadi intermedi che GRUB utilizza per accedere ai contenuti di diversi filesystem (ne esiste uno per tipo di filesystem, e sono supportati tutti i principali filesystem di Linux). Il file `device.map` contiene invece la lista dei dispositivi riconosciuti da GRUB in fase di installazione, e come questi vengono mappati nella notazione interna di GRUB, un esempio di questo file è il seguente:

/boot/grub/device.map
```
(fd0)   /dev/fd0
(hd0)   /dev/hda
```

che ci mostra il caso di una macchina su cui sono presenti un floppy ed un disco rigido.

I dispositivi vengono identificati da GRUB con un nome fra parentesi tonde, il floppy viene sempre identificato con `fd0` (nel caso ci siano due floppy ci sarebbe anche `fd1`) mentre i dischi vengono identificati, in ordine di rilevazione, con `hd0`, `hd1`, ecc. Si tenga presente che GRUB non distingue i nomi per i dischi IDE o SCSI, quindi anche questi ultimi verrebbero identificati con la sigla `hdN`, dove `N` è semplicemente un numero progressivo che dipende dall'ordine di rilevamento.

Il file `device.map` viene generato con il comando di installazione di GRUB, `grub-install`; questo prende come argomento il dispositivo su cui si vuole installare GRUB come *bootloader* (si tenga presente che in tal caso si andrà a sovrascrivere un eventuale altro *bootloader* presente su tale dispositivo). Il comando copia anche nella directory `/boot` tutti i file di GRUB, e gli si può dire di usare una radice diversa con l'opzione `--root-directory=DIR`. Il comando esegue anche una scansione dei dispositivi e crea `device.map`.

In teoria GRUB non ha un vero e proprio file di configurazione, infatti se avviato come *bootloader* mette a disposizione una shell da cui è possibile eseguire tutti i suoi comandi interni. Questa stessa shell è disponibile anche se lo si esegue come semplice programma in user space, lanciandolo con il comando `grub`. I comandi di GRUB sono molteplici ed oltre a quelli usati per impostare l'avvio dei vari kernel, ce ne sono altri da usare in modalità interattiva, come `find` e `cat` che come gli analoghi della shell sono quelli che forniscono rispettivamente la capacità di cercare i file e di visualizzarne il contenuto. La shell di GRUB inoltre offre funzionalità avanzate come l'auto-completamento di comandi e nomi dei file, ed anche un *help on line* con il comando `help`.

Ma al di là della possibilità di uso interattivo la modalità principale con cui viene usato GRUB è quella in cui si inseriscono tutti i comandi necessari in un apposito file, `menu.lst`, posto sempre sotto `/boot/grub`, così che questi vengano eseguiti automaticamente all'avvio. In

questo modo `menu.lst` diventa una sorta di file di configurazione, tanto che alcune distribuzioni installano sotto `/etc` il file `grub.conf` come link simbolico a questo file, ed i vari comandi di GRUB assumono il significato di direttive di configurazione.

Il vantaggio di usare `menu.lst` è che questo permette di creare anche un menu semi-grafico in formato testo dal quale scegliere quale kernel o altro sistema avviare, pertanto in genere all'installazione di GRUB come *bootloader* segue anche quella di un appropriato `menu.lst`. Il contenuto tipico di questo file è qualcosa del tipo:

menu.lst

```
## default num
# Set the default entry to the entry number NUM. Numbering starts from 0, and
# the entry number 0 is the default if the command is not used.
#
# You can specify 'saved' instead of a number. In this case, the default entry
# is the entry saved with the command 'savedefault'.
default         0

## timeout sec
# Set a timeout, in SEC seconds, before automatically booting the default entry
# (normally the first entry defined).
timeout         5

# Pretty colours
color cyan/blue white/blue

## password ['--md5'] passwd
# If used in the first section of a menu file, disable all interactive editing
# control (menu entry editor and command-line)  and entries protected by the
# command 'lock'
# e.g. password topsecret
#      password --md5 $1$gLhU0/$aW78kHK1QfV3P2b2znUoe/
# password topsecret

title           Debian GNU/Linux, kernel 2.6.18-4-686
root            (hd0,0)
kernel          /boot/vmlinuz-2.6.18-4-686 root=/dev/hda1 ro
initrd          /boot/initrd.img-2.6.18-4-686
savedefault

title           Debian GNU/Linux, kernel 2.6.18-4-686 (recovery mode)
root            (hd0,0)
kernel          /boot/vmlinuz-2.6.18-4-686 root=/dev/hda1 ro single
initrd          /boot/initrd.img-2.6.18-4-686
savedefault
```

e su Debian si può anche usare il comando `update-grub`, che effettua una ricerca dei kernel presenti in `/boot/` e crea automaticamente una lista con le relative voci in `menu.lst`.[40]

Le varie voci che compariranno nel menu iniziale sono introdotte dal comando `title`, che specifica la stringa con cui queste verranno mostrate, ad esso si associa in genere il comando `root`, che specifica quale disco e partizione usare come "*radice*" per cercare i file a cui si fa

[40] il comando di Debian usa anche alcuni dei commenti dentro `menu.lst` per generare parametri e valori usati nelle singole voci create automaticamente.

riferimento nei comandi seguenti. Questa viene indicata con la notazione di GRUB, in cui si indica fra parentesi tonde il dispositivo, seguito da una virgola e dal numero progressivo della partizione a partire da zero. Nel caso precedente allora si dice di cercare il kernel nella prima partizione del primo disco, in sostanza `/dev/hda1`.

Comando	**Significato**
`chainloader`	indica di lanciare il *bootloader* che si trova all'inizio della partizione impostata come radice, prende in genere un argomento (`+1` nel caso di Windows) che indica a quanti settori dall'inizio della partizione stessa si trova il secondo *bootloader*.
`default`	indica quale fra le voci del menu deve essere avviata di default, prende come argomento un numero progressivo che parte da 0.
`initrd`	indica il file da cui caricare l'immagine di un *ramdisk*.
`kernel`	indica il file da cui caricare in memoria il codice da eseguire come kernel all'avvio.
`makeactive`	marca come "*attiva*" la partizione indicata come radice, necessario quando questa è la partizione di avvio di Windows (usato in abbinamento con `chainloader`).
`password`	imposta una password che blocca l'accesso alla riga di comando durante l'avvio.
`root`	imposta il dispositivo (disco e partizione) da usare come radice per i nomi dei file usati dagli altri comandi.
`timeout`	indica quanto tempo, in secondi, deve passare prima che venga avviata la voce di default.
`title`	imposta una voce nel menu di avvio, e usa l'argomento seguente come titolo della stessa.

Tabella 5.23: Principali comandi usati nella configurazione di GRUB.

L'immagine del sistema operativo da caricare si specifica con la direttiva `kernel`, seguita dal pathname del file. Si tenga presente che questo è relativo alla partizione stessa (non esiste il concetto di radice in GRUB), per cui se ad esempio si è posto `/boot` su un'altra partizione non è più necessario specificarla nel nome del file. Nel caso di Linux al nome del file si devono far seguire le opzioni da passare al kernel all'avvio, fra cui occorre comunque specificare il dispositivo da montare come radice (nell'esempio precedente con `root=/dev/hda1`), se si usa un *ramdisk* si dovrà specificare anche il relativo file con il comando `initrd`.

Il comando `default` permette di stabilire quale immagine, nella lista di quelle dichiarate, deve essere lanciata se l'utente non interviene, dopo un tempo di attesa specificato con `timeout`. Inoltre è possibile proteggere l'accesso alla shell di GRUB, che di default è sempre accessibile, con la direttiva `password`, che oltre alla password in chiaro accetta come argomento anche la stessa in formato MD5, scritta nella stessa forma in cui viene espressa in `/etc/shadow`.

I principali comandi usati come direttive di configurazione all'interno di `menu.lst` sono riportati in tab. 5.23, una lista dei comandi interattivi come accennato può essere ottenuta con il comando `help`, mentre l'elenco completo delle funzionalità di GRUB è illustrato in dettaglio sia nel *GRUB-HOWTO* che nelle pagine info del programma, accessibili con `info grub`.

Benché GRUB sia ancora diffuso, il suo sviluppo è stato ormai completamente abbandonato in favore della suo successore GRUB2, che lo sta progressivamente soppiantando. Questo però non è una evoluzione del precedente, ma una completa riscrittura, cosa che ha consentito di superare

alcuni dei limiti originali, come quello di non saper gestire il RAID software, che richiedeva la necessità di installare GRUB separatamente in maniera manuale su entrambi i dischi.

Le principali nuove funzionalità fornite da GRUB2 sono quella di poter accedere al contenuto dei volumi logici di LVM, quella di gestire meccanismi di partizionamento diversi da quelli classici (con il supporto delle nuove partizioni GPT usate da UEFI), la disponibilità di una versione usabile con UEFI, la possibilità di essere utilizzato su piattaforme diverse da quella dei PC e la capacità di identificare direttamente i filesystem con il loro UUID (vedi sez. 5.1.3) invece che attraverso un file di dispositivo o l'indicazione di una partizione.

Il problema è che con la riscrittura le modalità di configurazione sono state completamente rivoluzionate, basandole su un sistema di moduli che forniscono le varie funzionalità e su un linguaggio di scripting (con tanto di istruzioni condizionali) che viene eseguito dal *bootloader* stesso. Se da una parte questo ha aumentato la flessibilità e le capacità di GRUB2, dall'altra ha reso il file di configurazione molto più complesso rispetto al precedente `menu.lst`, portandolo al limite dell'illeggibilità, per cui diventa molto più complicato fare delle correzioni alla configurazione durante l'avvio.

Nell'uso comune però questa maggiore complessità viene mascherata in quanto il file di configurazione non deve più essere modificato direttamente, ma generato attraverso il comando `grub-mkconfig`. Di default il comando stampa la configurazione generata sullo *standard output*, e si deve usare l'opzione `-o` per effettuare il salvataggio nella posizione standard in cui viene letto all'avvio, che è `/boot/grub/grub.cfg`.[41]

La configurazione viene costruita da `grub-mkconfig` in base alle istruzioni contenute in una serie di script di shell, che fan parte del pacchetto, installati sotto `/etc/grub.d`. Oltre a questi viene utilizzato il contenuto del file `/etc/default/grub`, che così viene a costituire il file su cui si opera normalmente per le principali modifiche che si possono apportare alla configurazione. Il file è un segmento di codice shell che assegna dei valori ad alcune variabili, in modo che questi vengano utilizzati dagli script di `/etc/grub.d`.

Di default `grub-mkconfig` crea per ogni immagine del kernel presente in `/boot` due voci nel menù di avvio, una per l'avvio ordinario, ed una per l'avvio in *single user mode* (vedi sez. 5.3.5) o modalità di recupero equivalente. Le voci sono ordinate sulla base al numero di versione dei kernel presenti, mettendo in cima alla lista la versione più recente. Le opzioni di avvio da passare al kernel vengono indicate con le variabili `GRUB_CMDLINE_LINUX` (per tutte le voci) e `GRUB_CMDLINE_LINUX_DEFAULT` (aggiunte alle voci relative all'avvio ordinario). Se sulla macchina sono presenti anche altri sistemi operativi vengono generate in coda alla lista ulteriori voci per poterli lanciare.

Il kernel che viene avviato di default viene indicato assegnando un valore alla variabile `DEFAULT`, che viene a costituire l'equivalente della direttiva `default` di GRUB, il valore di default è 0, che indica la prima voce, che nella generazione del file di configurazione viene associata all'avvio ordinario del kernel con la versione più recente. La variabile consente, oltre all'indicazione numerica della voce da avviare (contando a partire da zero), di specificare una stringa identificativa (quella che compare nel menù di avvio, corrispondente alla direttiva `menuentry` di `grub.cfg`) o il valore speciale `saved` per ripartire dall'ultima voce usata. Le altre variabili più rilevanti sono riportate in tab. 5.24, per i dettagli si consulti la documentazione di GRUB2, accessibile con `info grub`, nella sezione "*Configuration*".

[41] su Debian si può continuare ad usare anche `update-grub`, che semplicemente chiama `grub-mkconfig` passandogli l'opzione "`-o /boot/grub/grub.cfg`".

Variabile	**Significato**
GRUB_CMDLINE_LINUX	indica gli argomenti da inserire nella riga di comando di tutti i kernel, sia quelli per l'avvio ordinario che per l'avvio in *single user mode*.
GRUB_CMDLINE_LINUX_DEFAULT	indica gli argomenti da inserire nella riga di comando del kernel per un avvio ordinario, da aggiungere a quelli di GRUB_CMDLINE_LINUX.
GRUB_DEFAULT	indica quale fra le voci del menu deve essere avviata di default, prende come argomento un numero progressivo che parte da 0 (il default), oppure l'etichetta che identifica una voce del menu iniziale, o la parola chiave saved per indicare quello salvato in un precedente avvio (nel qual caso occorre anche definire GRUB_SAVEDEFAULT=true).
GRUB_DISABLE_RECOVERY	se impostata a true non genera per ogni kernel la seconda voce usata per avviare il sistema in *single user mode*.
GRUB_SERIAL_COMMAND	indica la direttiva di configurazione per la seriale da usare se si è indicato serial come valore di GRUB_TERMINAL, da specificare con una linea contenente la direttiva serial ed i rispettivi argomenti usati da grub.cfg, ad esempio: "serial --speed=38400 --unit=0 --word=8 --parity=no --stop=1"
GRUB_TERMINAL	indica il terminale con cui interagire con GRUB2 all'avvio della macchina, con valori come console (il default, che indica la console nativa della macchina), serial (per l'uso di una seriale da impostare con GRUB_SERIAL_COMMAND) ed altri a seconda dell'architettura hardware della macchina; si possono specificare più valori separati da spazi per utilizzare dispositivi multipli.
GRUB_TIMEOUT	indica quanto tempo, in secondi, deve passare prima che venga avviata la voce di default.

Tabella 5.24: Principali variabili di /etc/default/grub usate nella configurazione di GRUB2.

Si tenga presente che grub-mkconfig genera semplicemente il file di configurazione, per l'installazione è sempre disponibile il comando grub-install che opera in maniera analoga al vecchio GRUB e richiede come argomento il dispositivo su cui effettuare l'installazione, solo che in questo caso è in grado di funzionare correttamente anche se lo installa sui dischi di un RAID1 software (vedi sez. 6.1.3). Il comando inoltre supporta anche l'installazione su sistemi che usano EFI al posto del BIOS tradizionale, nel qual caso la partizione di sistema di EFI deve essere montata sotto /boot/efi e non sono necessari ulteriori argomenti.

In realtà in questo caso grub-install può dipendere da una corretta impostazione dei valori di device.map, e questo, se è cambiata la versione del kernel (ad esempio perché si sta usando un sistema di recupero), può contenere, essendo in genere basato su identificativi dei dischi generati da *udev*, dei nomi non corrispondenti a quelli del kernel corrente, che farebbero fallire il comando.

Il contenuto di questo file può comunque essere rigenerato con il comando grub-mkdevicemap, che lo aggiorna agli identificativi corretti secondo il kernel corrente, così che i dischi su cui installare GRUB2 possano essere identificati correttamente. Si può anche usare il comando con l'opzione -m seguita da un pathname per ottenere il risultato su un file diverso, o direttamente sullo *standard input* con il nome "-".

Benché grazie all'uso di grub-mkconfig, non sia più necessario intervenire manualmente su

grub.cfg, cosa da evitare sempre visto che le modifiche verrebbero perse alla successiva invocazione del comando,[42] una conoscenza di base di alcune direttive fondamentali utilizzate in questo file resta comunque necessaria qualora ci si debba trovare nella condizione di doverle modificare in fase di avvio dalla console di GRUB2. Non tratteremo pertanto in maniera estesa la sintassi di questo file di configurazione (che come accennato consente anche costrutti evoluti come condizionali e cicli) limitandoci a prendere in esame il necessario per orientarsi in caso di emergenza.

Direttiva	**Significato**
boot	esegue l'avvio del sistema operativo caricato; da usare esplicitamente solo sulla riga di comando, è implicita nella chiusura di una voce di menu.
initrd	carica l'immagine di un *ram disk* e ne predispone l'uso da parte del kernel; richiede come argomento obbligatorio il pathname a partire dalla radice indicata dalla variabile di ambiente root.
insmod	carica un modulo, da indicare come argomento, a partire dalla directory indicata dalla variabile di ambiente prefix (in genere /boot/grub).
linux	carica il file con l'immagine di un kernel Linux; richiede come primo argomento obbligatorio il pathname a partire dalla radice indicata dalla variabile di ambiente root, cui possono essere aggiunti eventuali argomenti successivi che saranno passati al kernel (quelli di cui abbiamo parlato in sez. 5.3.1).
menuentry	crea una voce per il menu di avvio, deve essere seguita da una stringa (delimitata dal carattere "'") che ne indica il titolo, da eventuali opzioni e dalla serie di direttive da eseguire per quella voce, comprese fra parentesi graffe.
serial	attiva la console su seriale, richiede che si indichi con --unit l'interfaccia seriale da usare (se più di una sono numerate progressivamente a partire da zero), con --speed la velocità, con --word per il numero di bit, con --parity l'uso della parità, con --stop gli *stop bit* (per il significato di questi parametri si rimanda a sez. 5.4.3).
set	imposta una variabile di ambiente di GRUB2, nella forma variabile=valore.

Tabella 5.25: Principali direttive di configurazione di GRUB2.

Dalla console di avvio infatti è sempre possibile, attraverso l'interfaccia fornita da GRUB2, modificare sia la configurazione generale che le varie direttive utilizzate all'interno di ciascuna voce di menu. Fra queste le due più rilevanti sono linux che indica il pathname assoluto all'immagine del kernel da avviare, seguita dagli argomenti della riga di comando con cui lo si lancia (quelli di sez. 5.3.1), e initrd che indica il pathname assoluto del *ramdisk* usato nell'avvio. Qualora si voglia far partire un kernel in una posizione diversa (ad esempio perché si è perso il contenuto di /boot) si dovrà intervenire su queste direttive. Come accennato una trattazione dettagliata di tutte le direttive va al di la dello scopo di questo testo, si sono elencate più rilevanti in tab.5.25, per il resto si rimanda alla documentazione accessibile con info grub.

[42]qualora si vogliano introdurre delle direttive permanenti, ad esempio per eseguire una configurazione ad hoc non gestibile attraverso /etc/default/grub, queste devono essere generate con un opportuno script dentro /etc/grub.d/; ad esempio Debian fornisce il file 40_custom dove mettere le proprie personalizzazioni.

Un'altra direttiva fondamentale molto usata nella configurazione è `set`, che in generale consente di impostare una variabile di ambiente nella forma `variabile=valore`. Si può definire e poi riutilizzare una variabile qualunque, ma GRUB2 usa alcune variabili di ambiente speciali (il cui nome è riservato) per modificare il proprio comportamento, ed è appunto attraverso l'impostazione di dette variabili che viene effettuata una parte essenziale della configurazione di `grub.cfg`.

Variabile	**Significato**
`default`	indica la voce di default da usare per l'avvio del sistema, può essere indicato con il valore numerico della stessa (contata da 0) o con il valore della stringa usata come argomento della direttiva `menuentry` che la definisce.
`prefix`	indica il pathname della directory dei file di GRUB2 (`/boot/grub`) da cui vengono risolti i nomi dei file usati da GRUB2 (ad esempio quelli dei moduli).
`root`	indica il filesystem da usare come radice per la risoluzione dei nomi dei file, equivalente alla direttiva `root` di GRUB1, il valore deve essere specificato con la sintassi illustrata in seguito.
`timeout`	indica il tempo da attendere (in secondi) prima di avviare il sistema con la voce di default.

Tabella 5.26: Principali variabili di ambiente della configurazione di GRUB2.

Fra queste variabili speciali sono di particolare rilevanza `root`, che serve ad indicare il dispositivo ed il filesystem su cui cercare i file usati da GRUB2, e `default`, che indica il kernel da avviare di default, e che in sostanza svolgono il ruolo delle omonime direttive di GRUB1. Le altre variabili di ambiente speciali più rilevanti sono illustrate in tab. 5.26, per l'elenco completo si consulti la sezione "`Environment`" della documentazione, accessibile con `info grub`.

Si tenga presente però che con GRUB2, avendo ottenuto il supporto di molteplici dispositivi, schemi di partizionamento e filesystem, è stata modificata la modalità di identificazione dei dispositivi, pertanto i valori da usare per la variabile di ambiente `root` non corrispondono necessariamente a quelli usati per la omonima direttiva di GRUB, anche se per compatibilità si possono usare gli stessi valori con lo stesso significato.

I dischi infatti vengono ancora identificati direttamente con la stessa nomenclatura e numerati progressivamente a partire da zero, ma in genere le partizioni sono identificate con una stringa che ne specifica il formato (che per quelle ordinarie è `msdos`) e viene mantenuta la numerazione ordinaria vista in sez. 5.1.2, pertanto la prima partizione del primo disco sarà identificata come `(hd0,msdos1)` e non come `(hd0,0)`. Se invece si usa il nuovo di schema di partizionamento con GPT si potrà indicare direttamente una partizione con il suffisso `gpt`, utilizzando una indicazione come `(hd0,gpt1)`.

Ma oltre alla identificazione diretta in stile classico, grazie all'uso degli opportuni moduli, GRUB2 è in grado di utilizzare direttamente i filesystem su un RAID software o su un volume logico di LVM (che tratteremo rispettivamente in sez. 6.1.3 e 6.2), ed è in grado di accedervi sia identificandoli direttamente (per numero di array o con la sintassi `nomegruppo-nomevolume`) che attraverso il relativo UUID (si rammenti sez. 5.1.3); pertanto si potranno avere indicazioni come `(md/0)`, `(lvm/`*`vg-root`*`)` o `(mduuid/`*`2e405c525b920c94e725973a2a998466`*`)`.

5.3.4 Lancio del kernel, *ramdisk* ed avvio del sistema

Come accennato in sez. 5.3.1 uno dei compiti del *bootloader* all'interno della procedura di avvio è quello di passare degli argomenti al kernel all'atto del metterlo in esecuzione. Infatti in fin dei conti anche il kernel un è programma per cui gli si possono passare degli argomenti, solo che normalmente non esiste una riga di comando da cui darglieli, se non quella che può eventualmente mettere a disposizione il *bootloader*, che in questo caso svolge il ruolo della shell, e deve anche essere in grado, per alcuni argomenti, di fornire una opportuna interpretazione.

Argomento	**Significato**
`console=dev`	specifica un file di dispositivo da usare come *console* per i messaggi del kernel.
`init=command`	specifica il programma da lanciare come primo processo del sistema (di default è `/sbin/init`).
`initrd=file`	specifica un *RAM disk* iniziale che verrà usato come radice provvisoria; in realtà questo parametro viene utilizzato solo dal *bootloader* che ha anche il compito di interpretare il significato di `file` per ottenere i dati del *ramdisk*.
`mem=size`	imposta la dimensione della memoria da utilizzare, veniva utilizzato per soprassedere il malfunzionamento del sistema di auto-rilevamento del kernel.
`noacpi`	non utilizza ACPI per l'instradamento degli *interrupt* di processore.
`noapic`	disabilita l'uso delle I/O APIC (vedi sez. 5.4.1).
`nosmp`	disabilita il funzionamento in modalità multiprocessore.[43]
`quiet`	disabilita la stampa sulla *console* della maggior parte dei messaggi del kernel.
`ro`	all'avvio monta la directory radice in sola lettura.
`root=dev`	specifica (in genere col nome del file di dispositivo) il disco (o la partizione) da montare come radice.
`rootdelay`	numero di secondi da attendere prima di tentare di montare la radice (utile per dischi lenti a partire).
`single`	indica ad `init` di andare in *single user mode* (vedi sez. 5.3.5).

Tabella 5.27: Principali argomenti per la linea di comando del kernel.

In genere gli argomenti passati al kernel servono ad impostare parametri per alcuni sottosistemi o ad attivare o disabilitare funzionalità specifiche all'avvio. Essi vengono espressi nella stessa forma dei parametri per i moduli illustrati in sez. 5.2.5, anche perché i moduli che supportano l'uso di parametri a cui assegnare un valore in fase di caricamento (vedi sez. 5.2.5), quando vengono compilati all'interno del kernel possono ricevere il suddetto valore in questo modo. Pertanto gli argomenti possono essere espressi sia come singola parola chiave, che nella forma `chiave=valore`, dove `valore` può essere un numero, una stringa, o una lista separata da virgole.

In tab. 5.27 si è riportato un elenco degli argomenti del kernel più rilevanti nell'uso comune, ed il relativo significato; si può trovare una lista più completa nel file `kernel-parameters.txt` della documentazione allegata ai sorgenti del kernel o nel *boot-prompt-HOWTO*. La lista è solo

[43] opzione di compatibilità che disabilita le I/O APIC (vedi sez. 5.4.1) oltre far funzionare il kernel in modalità mono-processore, è l'equivalente di `maxcpus=0`, valore particolare dell'opzione che a sua volta consente di specificare quanti processori usare su un sistema SMP; in genere, come `noapci` e `noapic` la si usa in caso di problemi di avvio dovuti a implementazioni difettose del kernel, del BIOS o della scheda madre.

parziale, e riguarda solo i parametri del kernel, quelli dei moduli e quelli relativi ad architetture specifiche sono riportati in altri file sempre sotto la directory `Documentation` allegata ai sorgenti, ma la documentazione talvolta è incompleta e poco aggiornata. Si tenga presente che il kernel ignora gli argomenti che non riconosce, e che questi vengono passati direttamente al programma di avvio, in questo modo è possibile usare gli stessi anche per controllare il comportamenti di quest'ultimo.

In alcuni casi può essere necessario usare degli argomenti di avvio diversi da quelli preimpostati; uno dei più comuni è quello in cui si deve usare l'argomento `root` per cambiare il dispositivo da usare come radice, ad esempio perché si è spostato il disco, o si vuole fare l'avvio da una partizione diversa, oppure occorre usare l'argomento `init` per specificare un programma di avvio diverso da `/sbin/init`, ad esempio per sostituirlo con `/bin/sh` facendo partire direttamente una shell di *root*, così da essere in grado di accedere alla macchina con privilegi amministrativi per ripristinare una password di *root* persa o dimenticata.

A parte questi due l'altro parametro più usato è `console` che consente di specificare il dispositivo relativo ad una *console* su cui dirigere i messaggi del kernel, che viene usato in particolare sui sistemi embedded per mandare i messaggi sulla linea seriale, a cui si potrà accedere con un modem. Una volta avviato il sistema è possibile risalire alla linea di comando con la quale è stato avviato il kernel, che viene riportata nel contenuto del file `/proc/cmdline`.

Infine può capitare, in particolare con alcune schede madri e BIOS difettosi, di dover disabilitare alcune funzionalità del kernel che altrimenti possono causare un blocco dell'avvio dello stesso a causa di alcuni bug; per questo motivo in certi casi può essere necessario far partire il kernel disabilitando alcune funzionalità, ad esempio disabilitando il sistema *APIC* (vedi sez. 5.4.1) per la gestione degli *interrupt*, o farlo partire in modalità mono-processore.

Come già detto fin da sez. 1.1.2, una volta messo in esecuzione, il kernel esegue due soli compiti: monta la directory radice e lancia il programma di inizializzazione del sistema, che tradizionalmente è `/sbin/init`, e tutto il resto viene demandato a quest'ultimo. Ovviamente per poter far questo il kernel deve essere in grado di accedere al dispositivo su cui si trova la radice, montare il relativo filesystem ed eseguire detto programma.

Due degli errori più comuni, che comportano l'impossibilità di proseguire nella procedura di avvio, sono proprio quelli relativi all'impossibilità di montare la radice, ad esempio perché ci si è dimenticati di inserire nel kernel il supporto per accedere al dispositivo su cui essa si trova o quello per il suo filesystem, o all'impossibilità di lanciare `init` ad esempio perché si è danneggiato il programma, o qualche libreria, o si è indicata come radice una partizione sbagliata.

Dato che oggi la varietà di dispositivi su cui è possibile installare un sistema è estremamente ampia, abbiamo visto in sez. 5.2.4 come nella gran parte dei casi (è una politica ormai adottata da tutte le distribuzioni) sia previsto di appoggiarsi all'uso di un *ramdisk* che contenga l'immagine di un filesystem iniziale (creato in RAM) che sarà quello effettivamente montato come radice.

In questa modalità di avvio si ha un ulteriore stadio in cui è il programma di avvio lanciato dal *ramdisk* che individua il supporto di installazione del sistema operativo; in tal caso infatti è possibile montare sia `/sys` che `/proc` ed utilizzare comandi come `blkid` per identificare i dispositivi con gli UUID, caricare tutti i moduli necessari, avviare il RAID e riconoscere i dispositivi LVM qualora questi fossero utilizzati. Alla fine si potrà utilizzare il valore dell'argomento `root` passato al kernel (rilevabile dal contenuto di `/proc/cmdline`) anche in forma diversa dell'indicazione di un file di dispositivo, ed usare supporti rilevabili o disponibili solo in un secondo tempo.

Una volta identificato il dispositivo che contiene la propria installazione del sistema operativo, il programma di avvio del *ramdisk* lo monta e poi utilizza il comando `pivot_root` per spostare su di esso la radice ed infine esegue il programma di inizializzazione finale al di sotto di questa al proprio posto con una `exec`; di nuovo qual'è questo programma potrà essere scelto sulla base dell'argomento `init` passato al kernel, sempre rilevabile dal contenuto di `/proc/cmdline`.

Dato il suo ruolo particolare, il primo processo lanciato dal kernel (che per questo ha PID 1) è in sostanza il protagonista principale della procedura di avvio e svolge un compito fondamentale per il funzionamento del sistema operativo, dato che è suo compito mettere in esecuzione tutti i programmi che lo rendono effettivamente utilizzabile;[44] per questo è a lui che si fa riferimento per la gestione dei servizi da avviare o fermare. E come per le altre componenti interessate al processo di avvio anche in questo caso esistono varie alternative (e diversi programmi) che possono svolgere questo compito.

Fino a qualche anno la principale differenza era quella originata negli anni '70, con la scissione con le varianti di Unix derivate dalla versione sviluppata alla AT/T, che poi diventerà *System V*, o quella sviluppata a Berkeley, che darà origine alla famiglia dei BSD. In questo caso la differenza non era tanto nelle funzionalità di `init`, dato si che tratta sempre di lanciare gli opportuni programmi, quanto nella modalità in cui questi venivano avviati.

Nei sistemi derivati da BSD questo veniva fatto attraverso l'esecuzione di alcuni script. Attivare o meno un servizio dipende dall'inserimento o meno (al posto "*giusto*") delle opportune righe di avvio all'interno di essi. I sistemi derivati da *System V* usano una procedura più complessa, con script suddivisi per servizio ed eseguiti in opportuna sequenza (che tratteremo in dettaglio in sez. 5.3.5) che offre maggiore funzionalità e soprattutto è più "*modulare*".

Nel caso di Linux, che è stato sviluppato da zero e non deriva da nessuna delle due famiglie di Unix, si è sempre cercato di prendere il meglio dei due mondi, il risultato è che pochissime distribuzioni hanno adottato la modalità di BSD, e la gran parte han preferito, per la sua maggiore flessibilità, lo stile di avvio di *System V*.

In realtà, anche se lanciato per primo, e con un ruolo particolare, `init` resta comunque un programma eseguito in *user space*, che può essere tranquillamente sostituito da altri (ad esempio da una shell, come accennato in precedenza), per cui negli anni sono state proposte varie alternative, anche per cercare di superare le problematiche relative alla intrinseca sequenzialità della procedura di avvio classica, che può portare a tempi di partenza del sistema anche piuttosto lunghi.

Questo ha dato vita alla realizzazione di una serie di alternative, come *Upstart*, *OpenRC*, `systemd`, ispirate a filosofie progettuali completamente diverse, che esamineremo in sez. 5.3.6, concentrandoci in particolare su `systemd`, che sta ottenendo una adozione molto ampia, essendo stato scelto come nuovo programma di avvio nelle versioni più recenti delle principali distribuzioni come Debian, RedHat, SuSE e derivate.

5.3.5 Il sistema di avvio alla *System V*

Come accennato in sez. 5.3.4 fino qualche anno fa la gran parte delle distribuzioni GNU/Linux aveva adottato lo stile di avvio di *System V*, basato un `init` tradizionale, che resta comunque supportato, per compatibilità, anche da tutti i candidati alla sua sostituzione. Per questo lo

[44] e non solo per questo, si ricordi il ruoli che `init` che svolge nella gestione dei processi *orfani* (vedi sez. 1.3.2).

tratteremo per primo anche se oggi la sua diffusione sembra essere destinata ad essere sempre meno rilevante.

L'avvio "*alla System V*" (o *SysV init*, come viene spesso chiamato) avviene sulla base dei cosiddetti *runlevel*, una sorta di *modalità operative* del sistema in cui viene lanciato un particolare insieme di programmi, che garantiscono la presenza di un certo gruppo di servizi. È possibile selezionare sia quali programmi lanciare (ed in che ordine) in ciascun *runlevel*, sia quale *runlevel* utilizzare. Inoltre è possibile passare da un *runlevel* all'altro. È compito di `init` portare il sistema in un certo *runlevel*, secondo quanto specificato nella sua configurazione.

I *runlevel* validi sono 8, quelli effettivamente utilizzati sono numerati da 0 a 6. A questi si aggiunge il runlevel "S", che riveste un ruolo speciale e serve ad indicare quali sono i servizi che devono essere attivati comunque. La standardizzazione di LSB prevede che tre runlevel siano riservati; il *runlevel* 0 viene usato per fermare il sistema, mentre il *runlevel* 6 per riavviarlo. Infine il *runlevel* 1 serve per andare nel cosiddetto *single user mode*, la modalità di recupero in cui può entrare nel sistema solo l'amministratore e solo dalla console, con tutti i servizi disattivati e con la directory radice montata in sola lettura.

Per gli altri *runlevel* le caratteristiche possono variare da distribuzione a distribuzione, con o senza servizi di rete, avvio terminante con il login direttamente da X, etc. Per Debian il default prevede che da 2 a 5 essi siano tutti equivalenti, mentre RedHat e SuSE usano il 2 per l'avvio in console senza rete, il 3 per l'avvio in console con la rete ed il 5 per l'avvio in modalità grafica.

Tutto il procedimento di avvio viene controllato dal file di configurazione di `init` che è `/etc/inittab`. Il formato di questo file è molto semplice, come per molti altri le linee vuote o che iniziano per "`#`" vengono ignorate, le altre prevedono la presenza di quattro campi separati dal carattere "`:`" nella forma:

id:*runlevels*:*action*:*process*

Il primo campo, *id*, è una semplice etichetta composta da 1 a 4 caratteri che identifica la singola linea (anche se in certi casi fa riferimento a delle azioni speciali). Il secondo campo, *runlevels*, indica la lista dei *runlevel* a cui si applica la riga in questione. Il terzo campo, *action*, indica la modalità con cui `init` eseguirà il comando indicato nel quarto campo, *process*, quando il *runlevel* corrente corrisponde ad uno di quelli elencati nel secondo campo. In questo modo è possibile utilizzare `inittab` per elencare ad `init` quali sono i programmi da lanciare all'avvio e quali sono le operazioni da eseguire per ciascun *runlevel*.

Per capire meglio il funzionamento della procedura di avvio di *SysV* conviene però fare riferimento ad un esempio del contenuto di `inittab`. Riportiamo di seguito un estratto dalla versione del file installata su una Debian Squeeze:

/etc/inittab

```
# The default runlevel.
id:2:initdefault:
# Boot-time system configuration/initialization script.
# This is run first except when booting in emergency (-b) mode.
si::sysinit:/etc/init.d/rcS
# What to do in single-user mode.
~~:S:wait:/sbin/sulogin
# /etc/init.d executes the S and K scripts upon change
# of runlevel.
l0:0:wait:/etc/init.d/rc 0
```

```
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
...
# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
# What to do when the power fails/returns.
pf::powerwait:/etc/init.d/powerfail start
pn::powerfailnow:/etc/init.d/powerfail now
po::powerokwait:/etc/init.d/powerfail stop
# /sbin/getty invocations for the runlevels.
#
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6
...
```

Si tenga presente che questo file viene letto da `init` all'avvio mentre le azioni in esso specificate vengono eseguite sia nella procedura di avvio che ad ogni cambio di *runlevel* (torneremo su questo argomento più avanti). Specificando come azione `wait` si richiede che il programma sia eseguito una sola volta, attendendo che esso sia concluso prima di passare all'azione successiva; nell'esempio è questo il caso con lo script `/etc/init.d/rc` che, come vedremo fra poco, è quello che viene usato per avviare e fermare i servizi. Se invece non è necessario attendere la conclusione si può usare `once`.

Specificando `respawn` si chiede che il comando sia messo in esecuzione immediatamente, senza attendere la sua conclusione, per proseguire coi successivi; si richiede inoltre che esso sia rilanciato automaticamente ogni volta che se ne termina l'esecuzione. Questo è quello che nell'esempio viene fatto con `getty` per avere i terminali di login attivi sulle varie console: ogni volta che ci si collega al sistema `getty` eseguirà `login` per l'autenticazione e questo eseguirà la shell,[45] all'uscita dalla shell, `init`, accorgendosi della terminazione del processo, rilancerà di nuovo `getty`.

Il problema che spesso si verifica nella interpretazione del contenuto di `inittab` è che il significato di alcune azioni non è di immediata comprensione, in quanto ve ne sono di indipendenti dal *runlevel* scelto, come `powerwait`, `powerfailnow`, `powerokwait`, mentre `initdefault` non esegue nulla ma serve invece ad impostare il *runlevel* di default, cioè quello che viene usato quando viene fatto partire il sistema. Inoltre non contribuisce alla chiarezza il fatto che il campo `id` può essere soggetto a delle restrizioni, come nel caso delle righe di `getty` che richiedono vi si usi il numero della console.

[45]si ricordi che mettere in esecuzione un nuovo programma non comporta la creazione di un nuovo processo, nella catena di esecuzioni successive il processo resterà sempre lo stesso.

Valore	Significato
`boot`	il comando viene eseguito una sola volta all'avvio del sistema; il valore del *runlevel* viene ignorato.
`bootwait`	il comando viene eseguito una sola volta all'avvio del sistema attendendo per il suo completamento; il valore del *runlevel* viene ignorato.
`ctrlaltdel`	esegue il processo specificato quando `init` riceve un segnale di `SIGINT` o se si utilizza la combinazione di tasti `ctrl-alt-del`, viene usato in genere per invocare `shutdown` (vedi sez. 5.3.7).
`initdefault`	indica quale *runlevel* deve essere utilizzato all'avvio, il campo indicante il programma da eseguire viene ignorato.
`once`	il comando viene eseguito una volta all'ingresso nel *runlevel* specificato senza aspettare la sua terminazione prima di proseguire.
`powerfailnow`	eseguito quando viene comunicato ad `init` da parte del programma di gestione del gruppo di continuità che le batterie si stanno esaurendo.
`powerokwait`	eseguito quando viene notificato ad `init` il ritorno della corrente sulla linea elettrica.
`powerwait`	eseguito quando viene notificata ad `init` la caduta della linea elettrica da parte del programma di gestione del gruppo di continuità.
`respawn`	il comando viene eseguito all'ingresso nel *runlevel* senza aspettare la sua terminazione e viene rieseguito tutte le volte che termina.
`sysinit`	il comando viene eseguito all'inizio della procedura di avvio, prima di qualunque voce relativa alle azioni `boot` o `bootwait`; il valore del *runlevel* viene ignorato.
`wait`	il comando viene eseguito una volta all'ingresso nel *runlevel* specificato e si aspetta la sua terminazione prima di proseguire.

Tabella 5.28: Principali valori delle azioni indicabili nel terzo campo di `/etc/inittab`.

In tab. 5.28 sono riportati i principali valori ed il rispettivo significato delle parole chiave che possono essere usate per il campo `action`; i dettagli e l'elenco completo si trovano al solito nella pagina di manuale, accessibile con `man inittab`.

In genere non c'è molto da fare con questo file, l'unica cosa che può servire è cambiare la linea dell'azione `initdefault` per cambiare il *runlevel* di default, e passare ad esempio dal login da *console* a quello grafico.[46] Una seconda azione che si può voler modificare (ad esempio disabilitare) è quella che imposta la reazione alla combinazione di tasti `ctrl-alt-del` che nell'estratto è specificata dall'azione `ctrlaltdel`, ed invoca il programma `shutdown` (vedi sez. 5.3.7) per riavviare il computer.

Si noti come nell'esempio citato il primo comando eseguito sia quello associato dall'azione `sysinit`. Questa è una delle azioni speciali indipendente dal *runlevel*, che viene eseguita soltanto all'avvio della macchina e che nell'esempio prevede l'esecuzione dello script `/etc/init.d/rcS`. Questa è la scelta di Debian in cui questo script non fa altro che chiamare lo stesso `/etc/init.d/rc` invocato sul *runlevel* S, in modo che si possono inserire nella directory `/etc/rcS.d` gli script che si

[46] sempre che questo sia previsto come un *runlevel* indipendente dalla distribuzione; per RedHat questo significa mettere un 5 al posto di 3, mentre in Debian questa differenza non esiste.

vuole siano eseguiti all'avvio indipendentemente da qualunque *runlevel*. Su RedHat (e derivate) viene invece usato al suo posto lo script `/etc/rc.d/rc.sysinit` in cui di nuovo sono inserite tutte le operazioni di prima inizializzazione del sistema, da eseguire qualunque sia il *runlevel* scelto.

Come accennato il grosso della procedura di avvio viene gestito dallo script `/etc/init.d/rc` che nell'esempio precedente viene invocato, sia pure con un argomento diverso, per ciascun *runlevel*, utilizzando l'azione `wait` per attenderne il completamento. È compito infatti di questo script fermare e avviare i servizi che sono richiesti per il *runlevel* specificato come argomento.

Per eseguire questo compito esso si basa sul fatto che le modalità con cui si ferma ed avvia un servizio sono state a loro volta standardizzate. In particolare secondo le specifiche LSB è previsto che ciascun programma e servizio debba fornire un proprio script di avvio, da porsi, secondo quanto specificato dal *Filesystem Hierarchy Standard*, in `/etc/init.d/`.[47] Questi script devono prevedere anche l'uso di alcuni argomenti standard: `start`, che avvia il servizio, `stop` che lo ferma, `restart` che prima lo ferma e poi lo riavvia, `reload` che fa rileggere la configurazione.

Se allora si va ad analizzare il codice di `/etc/init.d/rc`, che può essere un buon esercizio per studiare la sintassi avanzata della shell, ci si accorgerà che a grandi linee quello che lo script fa è verificare la presenza di una directory `/etc/rcN.d`, (alcune vecchie distribuzioni potrebbero mantenerla ancora sotto `/etc/init.d/`) dove *N* corrisponde al numero del *runlevel* passato come argomento, per ricavare la lista dei file in essa presenti. Osservando il contenuto di queste directory vedremo che contengono tutte una serie di link simbolici agli script di avvio dei servizi posti in `/etc/init.d/`, chiamati con lo stesso nome ma preceduto da una sigla composta da una `S` od una `K` seguite da un numero di due cifre.

Dopo aver letto la lista dei file presenti `/etc/init.d/rc` si limita ad eseguire tutti questi script in ordine alfabetico, passando l'argomento `stop` a quelli che iniziano per "`K`" (che sta per *kill*) e l'argomento `start` a quelli che iniziano per "`S`" (che sta per *start*). In questo modo i servizi indicati dai rispettivi script vengono fermati o avviati, e nell'ordine stabilito dalle due cifre usate nella sigla.

In questo modo è possibile, per ogni servizio, avere uno script di avvio personalizzato, da mettere in `/etc/init.d/`, e poi programmarne la partenza o l'arresto in un certo *runlevel* creando un semplice link simbolico nella relativa directory. Si può anche, grazie alle due cifre presenti nei nomi dei link, decidere in quale punto della sequenza di avvio lanciare un certo servizio, ad esempio un server di rete dovrà essere lanciato sempre dopo che questa è stata attivata.

Si noti inoltre come la presenza dei servizi attivi in un determinato *runlevel* possa dipendere da come si è arrivati ad esso; nel passaggio da un *runlevel* ad un altro infatti i servizi avviati dal primo vengono fermati dal secondo solo in presenza di un corrispondente script iniziante per `K`, altrimenti restano attivi, anche se non sono previsti fra quelli che verrebbero avviati se si andasse direttamente in quel *runlevel*.

Benché sia possibile attivare o disattivare un servizio su un certo *runlevel* creando semplicemente un link simbolico con l'opportuna sigla iniziale, sono comunque disponibili innumerevoli programmi in grado di automatizzare il compito. Alcuni di questi consentono anche di generare la corretta posizione nella sequenza di avvio, utilizzando le informazioni presenti negli script stessi (in genere in forma di commento iniziale).

---

[47] tradizionalmente gli script di avvio di RedHat erano posti invece sotto `/etc/rc.d`; questo non corrisponde alla standardizzazione di LSB, per cui nelle versioni più recenti di questa distribuzione le varie directory previste dallo standard sono state realizzate tramite dei link simbolici, alcuni pathname potrebbero comunque non corrispondere.

Nel caso di Debian e derivate gli script di avvio sono in genere gestiti con `update-rc.d`. Il comando prende come primo argomento il nome dello script di avvio, e come secondo l'azione da compiere su di esso. L'uso più comune del comando è con l'azione `remove`, che consente di rimuovere da tutti i *runlevel* ogni riferimento (sia di avvio che di arresto) allo script indicato. In genere, a meno che non si sia già cancellato lo script da `/etc/init.d/`, occorre anche usare l'opzione `-f`, altrimenti il comando fallirà, pertanto se si vuole rimuovere dalla procedura di avvio l'uso di `atd` si potrà eseguire il comando:

```
update-rc.d -f atd remove
```

Se invece si vuole installare uno script di avvio si potrà usare l'azione `defaults`, che utilizzando le informazioni presenti nello script stesso lo installerà (in avvio o in arresto a seconda di quanto richiesto) su tutti i *runlevel*. Se dette informazioni non sono presenti il comando lo installa in avvio sui *runlevel* 2, 3, 4 e 5, ed in arresto su 0, 1 e 6, con numero di sequenza pari a 20. Il comando consente anche di specificare singolarmente quale servizio avviare e fermare su quale *runlevel* ed a quale punto della sequenza, ma per i dettagli si rimanda alla lettura della relativa pagina di manuale.

La gestione dei link simbolici è stata comunque anche standardizzata all'interno di LSB, che prevede a questo scopo un altro comando, `chkconfig`, che è quello usato su RedHat e la maggior parte delle altre distribuzioni, ma presente anche su Debian, dove è disponibile un pacchetto omonimo. Il comando consente di elencare tutti i servizi disponibili con l'opzione `--list`, aggiungerne uno con `--add` o rimuoverlo con `--del`. Ad esempio su una RedHat si potrà rimuovere `atd` con:

```
chkconfig --del atd
```

Una delle differenze fra distribuzioni è anche quella relativa alla attivazione o meno di un servizio alla sua installazione, cosa che viene eseguita per Debian e derivate, e non viene fatta da RedHat e derivate. Per quest'ultima infatti è sempre richiesto che quando si vuole attivare un servizio all'avvio dopo averlo installato, si esegua il comando:

```
chkconfig --add service
```

Anche in questo caso il comando fa riferimento alle informazioni contenute negli script per determinare su quali *runlevel* e con quale numero di sequenza inserire gli script, si può comunque soprassedere il default specificando una lista di *runlevel* con l'opzione `--level`. Per i dettagli su tutte le opzioni anche in questo caso si faccia riferimento alla pagina di manuale.

Benché con il tempo la procedura di avvio classico appena illustrata sia stata standardizzata, e sia ormai possibile inserire una operazione specifica in un qualunque punto della sequenza di avvio, esistono alcuni file che storicamente venivano usati per effettuare delle configurazioni aggiuntive alla fine della sequenza stessa. In particolare un file ancora utilizzato è `/etc/rc.local`. Esso viene eseguito alla fine della procedura di avvio,[48] ed assume un po' il significato di quello che era l'`autoexec.bat` del DOS: contiene i comandi che si vogliono eventualmente dare dopo che tutti i servizi sono partiti, senza doversi preoccupare di doverli inserire opportunamente all'interno della sequenza di avvio.

[48] ad esempio nel caso di Debian esso viene eseguito alla fine di tutti i *runlevel* tranne il *single user mode*.

Una modalità alternativa, supportata ad esempio da Debian, è quella dell'uso della directory `/etc/rc.boot` nella quale si possono mettere gli script che si vuole siano eseguiti alla fine della procedura di inizializzazione. In tal caso occorre tenere presente che i file dovranno essere eseguibili, e il loro nome non dovrà contenere caratteri speciali; Debian infatti esegue gli script usando lo speciale comando `run-parts` che, come visto in sez. 3.2.1, lancia tutti gli script presenti in una directory posto che il loro nome sia nel formato adatto.

Entrambi questi metodi di personalizzazione delle operazioni da compiere dopo l'avvio sono comunque sempre meno utilizzati, la standardizzazione del sistema di avvio prevede che si utilizzi direttamente un apposito script da inserire nel *runlevel* opportuno.

Infine occorre tenere presente che gli script di avvio usati dal sistema di avvio possono essere usati anche direttamente quando si debba fermare, avviare o riavviare un servizio, o anche richiedere la rilettura della configurazione. In tal caso gli script possono essere invocati direttamente lanciandoli per pathname con gli argomenti standard riassunti in tab. 5.29.

Argomento	**Significato**
`start`	avvia il servizio.
`stop`	ferma il servizio.
`restart`	riavvia (ferma e poi avvia) il servizio.
`reload`	ricarica la configurazione (senza fermare il servizio).
`force-stop`	forza lo stop del servizio.
`force-reload`	forza la rilettura della configurazione anche se questo comporta il riavvio del servizio.

Tabella 5.29: Argomenti standard per gli script di avvio alla *System V*.

Finora abbiamo parlato delle modalità in cui avviene l'avvio ed in cui il sistema si porta in un certo *runlevel*, vedremo ora come è possibile cambiare *runlevel* e istruire il sistema, anche una volta avviato, a passare da un *runlevel* ad un altro.

Una delle modalità in cui si può modificare il *runlevel* su cui una macchina viene avviata è quella che prevede il passaggio tramite il *bootloader* delle opportune opzioni di avvio per il kernel (quelle illustrate in sez. 5.3.1). Alcune di queste infatti vengono passate direttamente ad `init`, in particolare l'indicazione di un semplice valore numerico porta nel relativo *runlevel*, ed è inoltre possibile usare l'opzione `single` per avviarsi in *single user mode*. Ovviamente questo tipo di indicazione, fintanto che non viene impostata in maniera permanente nella configurazione del *bootloader*, vale soltanto per il singolo avvio, ed è questo il motivo per cui nella configurazione di GRUB vista in sez. 5.3.3 è presente una voce separata per l'avvio in *single user mode*.

Nelle operazioni ordinarie, una volta arrivati sul *runlevel* di default, normalmente non c'è bisogno di passare dall'uno all'altro. Può capitare però di essere partiti in *single user mode*, o di esservi arrivati a causa di problemi nella fase di avvio, e che una volta che si siano completati i compiti amministrativi o risolto il problema, si voglia passare ad un *runlevel* ordinario. Oppure, in seguito all'insorgere di problemi, ci si può voler portare in *single user mode*.

In tal caso il comando usato per cambiare *runlevel* è `telinit` (che è un link allo stesso `init`) per cui si può usare anche direttamente quest'ultimo per compiere le stesse operazioni. Il comando prende come argomento il numero di *runlevel* su cui ci si vuole portare, e semplicemente notifica ad `init` (inteso qui come il processo in esecuzione con PID 1) la richiesta di cambiamento di *runlevel* (allo scopo `init` usa la pipe `/dev/initctl` come dispositivo di controllo). A questo punto sarà compito di `init` registrare in `/var/log/wtmp` il cambiamento di *runlevel* (vedi

sez. 2.4.4) e far eseguire prima gli script di arresto e poi quelli di avvio secondo quanto previsto per il *runlevel* scelto, a condizione ovviamente che i relativi servizi siano presenti nel *runlevel* di partenza: non verranno cioè fermati servizi non attivi, ne fatti partire servizi già attivi.

Oltre al valore numerico del *runlevel* di destinazione si possono passare a `telinit` altri argomenti; ad esempio con "`q`" si indica ad `init` di riesaminare `/etc/inittab` mentre con "`s`" si richiede di andare in *single user mode* (le lettere possono essere indifferentemente maiuscole o minuscole). Il comando può essere anche invocato, utilizzando le opportune opzioni, per modificare semplicemente il comportamento di `init`; in particolare con l'opzione `-t` si può indicare un tempo diverso, rispetto ai 5 secondi di default, fra l'invio del segnale `SIGINT` e quello del segnale `SIGKILL` quando viene eseguito l'arresto della macchina.

Variabile	**Significato**
`CONSOLE`	la *console* di sistema, il default è `/dev/console`.
`INIT_VERSION`	la versione di `init`.
`PATH`	le directory dei comandi, come illustrato in sez. 2.1.3, inizializzata a `/bin:/usr/bin:/sbin:/usr/sbin`.
`PREVLEVEL`	il *runlevel* precedente.
`RUNLEVEL`	il *runlevel* corrente.

Tabella 5.30: Variabili di ambiente definite da `init` per i suoi figli.

È possibile inoltre cambiare i valori delle variabili di ambiente che `init` definisce per i processi che lancia o definirne di nuove con l'opzione `-e`, seguita dall'assegnazione delle variabili stesse. In generale infatti quando `init` esegue una delle azioni specificate in `inittab` definisce anche per il processo eseguito le variabili di ambiente riportate in tab. 5.30.

5.3.6 Sistemi di avvio alternativi e `systemd`

Benché sia in uso da moltissimi anni, il sistema di avvio alla *System V* presenta numerosi problemi. Quello che viene considerato più rilevante, specialmente in materia di prestazioni, è che la partenza del sistema può risultare anche molto lenta a causa della serializzazione del lancio degli script di avvio. Cosa che ad esempio comporta si debba aspettare che siano partiti tutti i demoni di rete per avere una console, dato che, come risulta dall'esempio di `inittab` illustrato in sez. 5.3.5, `getty` viene lanciato per ultimo.

Inoltre, dato che il sistema fa partire comunque i servizi registrati su un *runlevel*, ne verrà tentato l'avvio comunque, anche quando potrebbe essere già noto in una fase precedente che questo fallirà. Ad esempio se la rete non funziona o non può essere configurata, verrà comunque tentato l'avvio dei servizi di rete, anche se è chiaro che non potranno mai funzionare. E se si fanno errori nel piazzare un servizio nella sequenza di avvio, perché magari lo si lancia prima di un altro da cui dipende, verrà comunque avviato e si dovranno subire le conseguenze dell'errore.

Infine l'utilizzo degli script di avvio, per quanto offra una grande flessibilità, comporta anche, per lo stesso motivo, il fiorire di una varietà di opzioni, scelte specifiche, default registrati nei file più vari, casi particolari da gestire a seconda delle varie distribuzioni o delle varie implementazioni di sistema su cui il servizio viene installato. Il tutto porta a complessità degli script sempre crescenti, soprattutto in termini di manutenzione degli stessi.

Per questo motivo negli ultimi anni si è iniziato a cercare una alternativa ad `init` che cercasse di risolvere questi ed altri problemi. Un primo passo è stato quello di inserire negli script stessi

un meccanismo di tracciamento delle dipendenze che quantomeno consentisse di creare una generazione corretta della sequenza degli script e la possibilità di lanciare in parallelo quelli per cui è possibile.

Ma oltre a questo sono stati anche realizzati dei programmi in grado di sostituire `init` e migliorare la qualità e la velocità del processo di avvio. Il primo tentativo serio di sostituzione è quello effettuato con la creazione di *Upstart* introdotto da Ubuntu ed usato come default fino alla versione 14.10, ed adottato anche per RedHat 6.0.

La caratteristica principale di *Upstart* è quella di poter realizzare l'avvio del sistema in maniera asincrona e parallela, e gestire in maniera esplicita le dipendenze fra servizi. Il programma infatti ha una architettura *event driven* in cui cioè viene *guidato* dagli eventi che avvengono nel sistema, il principale dei quali (denominato `startup`) è associato all'avvio del sistema. A ciascun evento possono essere associati dei compiti (*job*) da eseguire in risposta allo stesso, che a loro volta possono generare altri eventi che comporteranno ulteriori reazioni.

In questo modo si possono gestire le dipendenze avviando un servizio che dipende da un altro con un *job* attivato in risposta ad un evento generato da quest'ultimo. Tutto quello che non è associato ad un evento specifico potrà essere eseguito immediatamente, e tutti i *job* potranno comunque essere eseguiti in parallelo una volta innescati dai relativi eventi.

Questo significa ad esempio che è possibile associare l'esecuzione degli script di configurazione di una interfaccia di rete alla rilevazione della presenza del link sulla stessa, ed in questo caso si può anche evitare di lanciare i servizi di rete fintanto che questa non è disponibile, ed anche definire le dipendenze fra diversi servizi di rete (ad esempio avviare la posta dopo il DNS) in modo che questi partano nella giusta sequenza e solo quando tutti i prerequisiti sono soddisfatti.

Benché *Upstart* consenta di gestire il lancio in parallelo dei programmi e l'esecuzione degli stessi solo quando necessario, resta comunque un problema generale di serializzazione nell'avvio dovuto alle dipendenze, per cui alcuni programmi possono essere lanciati (vale a dire letti dal disco, portati in memoria a poi eseguiti) solo dopo che è stato completato l'avvio di quelli da cui dipendono.

Questo comporta un calo delle prestazioni in quanto se li si potessero lanciare in contemporanea si potrebbe ottenere una maggiore velocità concentrando gli accessi a disco ed il caricamento in memoria, ma per l'esistenza delle dipendenze si è costretti a serializzare gli avvii. Se però si vanno ad esaminare le ragioni per cui queste dipendenze esistono, ci si rende conto che nella gran parte dei casi esse sono dovute al fatto che un programma deve poter contattare un altro per uno scambio di dati, e che questo avviene nella quasi totalità dei casi attraverso un socket (o locale o di rete).

La considerazione da cui nasce `systemd` è che se questi socket fossero immediatamente disponibili non sarebbe necessario aspettare l'avvio del programma che deve crearli perché un secondo programma, che deve contattare il primo da cui dipende, possa essere lanciato. Se il socket che serve è già disponibile infatti il secondo programma può comunque partire ed aprirlo (che spesso è tutto quel che serve) per poi proseguire, se invece deve inviare dei dati può scriverli sul socket, o tentare una lettura se deve riceverli.

In questo scenario però si possono sfruttare le caratteristiche dell'interfaccia di programmazione dei socket. Questa prevede che le scritture siano bufferizzate per cui se non si eccedono le dimensioni dei buffer l'invio di dati ha successo immediato e l'esecuzione può proseguire. Se invece serve leggere una risposta dal socket in assenza di dati disponibili l'esecuzione si bloccherà, ma questo avverrà solo per quei programmi che devono eseguire una lettura, solo dopo che sono

partiti, senza generare errori e nell'esatto momento in cui l'esigenza si presenta. E l'esecuzione ripartirà immediatamente senza nessun intervento esterno non appena arriveranno i dati a cura di chi li deve fornire, quando questi inizierà a scriverli sul socket.

Si noti come utilizzando questo meccanismo non serve più doversi preoccupare di analizzare le dipendenze reciproche per definire una corretta sequenza di avvio, si può lanciare tutto in parallelo e le eventuali dipendenze verranno gestite automaticamente dall'interfaccia standard dei socket del kernel, che provvederà a fermare i processi che necessitano di dati, il tutto senza nessuna necessità di doverle identificare e gestire esplicitamente.

Questa modalità di avvio viene detta *socket activation* ed è una delle caratteristiche più importanti di `systemd`. In sostanza è `systemd` che si incarica di creare, sulla base della sua configurazione, tutti i socket necessari ai programmi che deve lanciare, passando loro i relativi file descriptor una volta che questi si sono avviati. In realtà il concetto è tutt'altro che nuovo: come vedremo in sez. 8.1.1 con i cosiddetti *superdeamon* dei servizi di rete, l'idea di un programma di supervisione che si pone in ascolto passando poi, in caso di connessione, quest'ultima al programma appropriato, esiste fin dagli albori di Unix. Lo stesso concetto è anche quello che viene utilizzato dal programma di avvio (`launchd`) usato da MacOsX.

Ovviamente per far funzionare la *socket activation* è necessario il supporto da parte dei programmi che usano i socket, che devono prevedere la possibilità di utilizzare quelli creati da altri invece di crearli in proprio. Questo comunque non si è dimostrato un grosso ostacolo, specie per i servizi di rete, dato che come accennato già esisteva un concetto affine con i *super-demoni*, per cui il supporto fornito da `systemd` si è rivelato abbastanza semplice da utilizzare, e laddove assente è stato rapidamente introdotto. A lungo termine questo consente anche di rendere i suddetti programmi indipendenti da tutta la logica relativa alla creazione di detti socket, essendo questa demandata a `systemd`.

Ma oltre all'uso della *socket activation* il concetto di mettere in esecuzione i programmi da lanciare in maniera asincrona rispetto alle risorse che devono utilizzare è stato esteso da `systemd` nella maniera più ampia possibile, in tutti quei casi in cui fosse possibile realizzarlo. Ad esempio è stato sfruttato l'*automounter* (vedi sez. 5.1.6) per eseguire immediatamente un programma anche quando questo richiede un accesso a dati posti su un disco non ancora montato. Invece di fermarsi ad aspettare che la scansione di `/etc/fstab`, con eventuale relativo controllo del filesystem e abilitazione delle quote, sia stata eseguita per tutti i filesystem, il programma potrà partire immediatamente e si bloccherà in attesa del completamento delle operazioni dell'*automounter*, ma solo lui, senza fermare una sequenza di avvio, e solo quando richiederà di accedere ad un file su un filesystem non ancora montato.

Un secondo aspetto di `systemd` che consente di migliorare le prestazioni di avvio è quello di evitare di lanciare tutto quello che non serve fintanto che non ne viene richiesto l'uso. Ad esempio è inutile lanciare `cupsd` fintanto che non viene richiesta una stampa, o avviare un servizio di rete se ancora l'interfaccia non è stata configurata, per cui `systemd` consente, appoggiandosi ad infrastrutture esistenti come quella di *udev* (vedi sez. 5.4.5), di lanciare automaticamente i programmi che lo utilizzano solo alla comparsa di un certo dispositivo, o come i *super-demoni*, fare partire i demoni di rete solo alla ricezione di una connessione.

Inoltre `systemd` supporta anche la possibilità di indicare esplicitamente dipendenze e conflitti, ottenendo un meccanismo simile a quello di *Upstart*, ed anche di forzare una eventuale serializzazione. Ed ovviamente resta in grado di usare il classico *SysV init*, trattando `/etc/init.d/` come una ulteriore fonte di configurazione, cercando comunque di ottimizzare le prestazioni usando le

eventuali informazioni di dipendenza contenute negli script di avvio e ricorrendo altrimenti alla struttura dei link nelle varie directory `/etc/rcN.d`.

Infine è stato fatto un notevole sforzo per sostituire il più possibile l'uso degli script di shell, ed al di là della possibilità di lanciare direttamente i programmi, `systemd` è in grado di effettuare internamente una serie di configurazioni di avvio, per cui ad esempio non è più necessario eseguire esplicitamente comandi appositi per impostare l'*hostname* della macchina, o i parametri del kernel (quelli di sez. 3.1.3).

Ma le novità introdotte da `systemd` vanno oltre gli aspetti collegati all'avvio dei processi. Il programma infatti si distingue anche per le sue funzionalità relative alla gestione ed al tracciamento degli stessi. Un'altra delle sue caratteristiche principali infatti è quella di un uso esteso dei *Control Group*, una funzionalità presente nei kernel più recenti, che gli consente di raggruppare tutti i processi associati ad un servizio che mette in esecuzione, sia per identificarli, che per tenerli sotto controllo o terminarli, che per applicare eventuali limiti sulle risorse all'intero gruppo, o impostarne priorità e privilegi.

Infine un'ulteriore importante funzionalità di `systemd`, cui abbiamo accennato in sez. 3.2.3, è quella di fornire anche un sistema di registrazione degli eventi di sistema che si pone come alternativa al *syslog*, realizzato tramite un programma di ausilio (`journald`) che a fianco di scelte molto discutibili (e discusse) come quella di fare uso di file binari il cui formato non è definito in maniera stabile, consente un notevole dettaglio e fornisce funzionalità di ricerca e classificazione delle informazioni molto interessanti.

Il file di configurazione principale del programma è `/etc/systemd/system.conf`, che però si applica solo al programma quando questo viene eseguito con PID 1, qualora lo si esegua dalla riga di comando a sistema avviato (quindi con un PID diverso da 1) viene invece usato il file `/etc/systemd/user.conf`. Per tutti i file di configurazione `systemd` ha adottato una variante del formato dei *keyfile* (illustrati in sez. 3.1.1) la cui principale estensione è che l'assegnazione multipla di valori diversi alla stessa chiave non ne sovrascrive il valore ma lo trasforma in una lista, mentre per cancellare un precedente valore e poi poterlo riassegnare occorre effettuare prima una assegnazione vuota.

Il file di configurazione generale permette anzitutto di definire alcune caratteristiche di funzionamento del programma; ad esempio con `LogLevel` si indica il livello di logging e con `LogTarget` la modalità di registrazione dei messaggi. Consente inoltre di impostare valori per risorse, limiti o proprietà generali che verranno assegnate a tutti programmi messi in esecuzione; ad esempio assegnando un valore a `DefaultEnvironment` nella forma di una stringa che definisce una lista di variabili di ambiente, queste verranno propagate a tutti i programmi che vengono avviati. Normalmente non esiste nessun motivo per modificare il default, e questi due file contengono soltanto direttive commentate.

Dato che come accennato `systemd`, compie direttamente alcune operazioni eseguite nell'avvio tradizionale di *SysV init* tramite l'uso di comandi esterni, i file che venivano utilizzati a tale scopo possono essere considerati come file di configurazione "*indiretti*" per `systemd`. Fra questi c'è `/etc/hostname`, usato per impostare il nome della macchina (vedi sez. 7.4.1) i file `/etc/sysctl.d/*.conf` usati per impostare i parametri del kernel (vedi sez. 3.1.3), `/etc/fstab`, usato per montare i filesystem ed attivare la *swap*, i file `/etc/modules-load.d/*.conf` usati per caricare moduli di kernel aggiuntivi all'avvio.

La gran parte della gestione dell'avvio con `systemd` fa riferimento al concetto di "*unità*", che costituisce una generalizzazione di quello che è il concetto di servizio nell'avvio tradizionale

alla *SysV*. Tutte le operazioni effettuate da `systemd` per avviare il sistema fanno riferimento alle unità che vengono gestite tramite i cosiddetti *unit file*. Sono questi file che controllano il comportamento del programma e vanno a sostituire gli script di avvio, estendendone il significato. Ad esempio anche le operazioni di configurazione legate ai file di configurazione di sistema elencati prima sono effettuate convertendo internamente gli stessi in opportune "*unità*".

Ciascuna *unità* è sempre associata ad un *nome* e ad un *tipo* e viene espressa nella forma *`nome.tipo`* che è anche il nome del relativo *unit file*. Gli *unit file* di sistema vengono installati (dai pacchetti della propria distribuzione) in `/lib/systemd/system/`, ne possono poi essere creati ulteriori in fase di esecuzione in `/run/systemd/system/`, mentre l'amministratore di sistema può effettuare le proprie configurazioni inserendole sotto `/etc/systemd/system/`. Queste tre directory vengono esaminate nell'ordine appena illustrato, caricando tutti gli *unit file* presenti; in caso siano presenti più *unit file* relativi ad una stessa unità viene utilizzata la configurazione presente nell'ultimo che viene letto. Per quanto appena detto questo fa si che le configurazioni sotto `/etc/systemd/system/` abbiano la precedenza sulle altre.

Infine se un *unit file* è vuoto, o se è un link simbolico a `/dev/null`, la relativa unità viene marcata come *mascherata* rendendone impossibile l'attivazione anche per via manuale. Questo consente di disabilitare permanentemente una unità semplicemente creando un link simbolico a `/dev/null` dentro `/etc/systemd/system/`, il cui contenuto, essendo letto per ultimo, ha la precedenza sulle altre configurazioni.

Quando `systemd` parte attiva l'unità `default.target` che provoca l'avvio del sistema (come vedremo le unità di tipo `target` servono a definire uno stato in cui si vuole portare il sistema). Questa a seconda dell'installazione effettuata è un link simbolico a `grafical.target`, se si è installata una stazione di lavoro con l'interfaccia grafica, o a `multi-user.target` per una installazione senza interfaccia grafica.

Questo default può essere modificato passando nella riga di comando del kernel (si ricordi quanto visto in sez. 5.3.4) l'argomento `systemd.unit=`*`unit.type`* per indicarne un altra. Ad esempio `rescue.target` è l'equivalente del *single user mode* ed un link simbolico a `runlevel1.target` (sono disponibili anche le unità `runlevel`*`N`*`.target` che riproducono gli altri *runlevel* classici), mentre `emergency.target` si limita ad eseguire una shell sulla console principale e si può usare al posto di `init=/bin/sh`, col vantaggio di poter proseguire con la procedura di avvio.

Argomento	**Significato**
`systemd.unit`	indica l'unità da attivare all'avvio (il default è `default.target`).
`systemd.crash_chvt`	con un valore positivo in caso di crash attiva il terminale virtuale indicato, il default è -1.
`systemd.crash_shell`	impostato a `true` indica di eseguire una shell in caso di crash di `systemd` (il default è `false`).
`systemd.confirm_spawn`	impostato a `true` richiede una conferma prima di avviare un qualunque processo (il default è `false`).

Tabella 5.31: Principali argomenti specifici di `systemd` da usare nella riga di comando del kernel.

Si sono riportati in tab. 5.31 i più significativi degli argomenti della linea di comando del kernel specifici di `systemd`, che sono sempre nella forma `systemd.`*`nome=valore`*. Ad essi si aggiungono anche i tradizionali valori usati per l'avvio classico, come `single` per il *single user mode* (equivalente a `rescue.target`) ed i vari numeri *`N`* usati per indicare un *runlevel*, equivalenti all'avvio con l'unità `runlevel`*`N`*`.target`.

Anche per gli *unit file* viene usato il formato dei *keyfile* con le estensioni precedentemente citate, i dettagli sono comunque illustrati nella documentazione accessibile con `man systemd.unit`. Dato che il nome serve ad identificare il soggetto dell'unità in generale può assumere un valore qualunque, anche se alcuni nomi, come `default.target` e gli altri citati sopra, sono riservati (per l'elenco completo si consulti la documentazione, accessibile con `man systemd.special`).

Il tipo invece serve a classificare le varie tipologie di operazioni che `systemd` può compiere attivando l'unità, e può assumere solo dei valori prestabiliti. I tipi più rilevanti, corrispondenti alle principali funzionalità del programma (per l'elenco completo si consulti `man systemd`), sono i seguenti:

`device` un dispositivo, da usare per eseguire delle operazioni in base alla disponibilità di un dispositivo;

`mount` un *mount point*, per controllare il montaggio dei filesystem con `systemd`;

`service` un servizio, per controllare l'avvio di demoni e la gestione dei relativi processi, sostanzialmente sostituisce uno script di avvio;

`socket` un socket, da usare per la *socket activation*, ogni unità di questo tipo deve avere una corrispondente unità di tipo `service` che viene fatta partire quando il socket riceve una connessione;

`target` un obiettivo, da usare per raggruppare altre unità e riottenere uno stato del sistema equivalente ad un *runlevel*; con questo tipo di unità si controlla quali programmi lanciare nella procedura di avvio.

La configurazione di una unità avviene definendo le opportune direttive in una opportuna sezione all'interno del suo *unit file*. Due sezioni `[Unit]` e `[Install]`, sono generiche e disponibili per qualunque tipo di *unit file*, ciascun tipo prevede inoltre una sezione omonima che contenga le sue direttive specifiche, ad esempio la sezione `[Service]` per i `.service`, la sezione `[Socket]` per i `.socket`, e così via.

La sezione `[Unit]` è la più rilevante nella gestione delle procedure di avvio e serve a mantenere le informazioni relative all'unità, in particolare consente di definirne le dipendenze da altre unità. Le due direttive principali per fare ciò sono `Require` e `Wants` che prendono come argomenti una lista di altre unità, espressa con i nomi dei relativi *unit file* separati da spazi o con la ripetizione della direttiva. Con `Require` sui indicano quali sono le altre unità "*richieste*" per l'attivazione di una unità; questo significa che quando si attiva l'unità, anche quelle indicate con `Require` verranno attivate, e viceversa se una di quelle indicate con `Require` viene disattivata, o la sua attivazione fallisce, verrà disattivato o fallirà anche l'attivazione dell'unità stessa.

Con `Wants` si indica invece una dipendenza più debole, in cui l'attivazione dell'unità comporta anche quella delle altre unità "*volute*", ma la disattivazione di una di queste, o il fallimento nella loro attivazione, non comporterà anche la disattivazione o il fallimento dell'unità stessa. Questo tipo di dipendenza viene in genere utilizzato quando si vuole associare l'attivazione di una unità a quella di un altra, e lo si fa semplicemente inserendola fra quelle "*volute*" da quest'ultima.

Data la loro rilevanza queste due direttive possono essere anche espresse in maniera diversa tramite l'uso di una eventuale directory nella forma *`unita.tipo`*`.require` o *`unita.tipo`*`.wants` in cui inserire altrettanti link simbolici agli *unit file* richiesti. La presenza di un link siffatto fa si

che la corrispondente unità sia implicitamente aggiunta a quelle che si sarebbero elencate con `Require` o `Wants`, e può essere gestita direttamente all'interno delle unità stesse attraverso l'uso delle direttive `RequiredBy` e `WantedBy` della sezione `[Install]`, che tratteremo a breve.

Si tenga presente che le dipendenze indicate da `Require` o `Wants` sono puramente logiche e non vanno intese nel senso tradizionale in cui un servizio deve partire prima di un altro; non hanno cioè nessuna influenza nell'ordine in cui vengono lanciati i servizi, cosa che `systemd` cerca sempre di fare in parallelo, usando le varie metodologie illustrate in precedenza. Qualora sia necessario definire una sequenza in cui un servizio deve essere avviato strettamente prima o dopo un altro, occorre dichiararlo esplicitamente usando rispettivamente le direttive `Before` o `After`. Nel primo caso `systemd` attenderà che l'unità abbia completato l'avvio prima di avviare quelle elencate, nel secondo caso la avvierà solo dopo che quelle elencate hanno completato il loro avvio. Come logico, visto il requisito, questo comportamento viene rovesciato in caso di stop.

Direttiva	**Significato**
`After`	richiede che l'unità sia attivata dopo il completamento dell'avvio di quella/e indicate.
`AllowIsolate`	deve essere impostata a `true` per poter utilizzare l'unità con il comando `isolate` di `systemctl` (ad uso prevalente delle unità `target`).
`Before`	richiede che l'avvio dell'unità si sia completato prima di attivare la/e unità indicate.
`Conflicts`	indica un conflitto con la/e unità indicate, l'attivazione di una di esse comporta la disattivazione dell'unità e l'attivazione dell'unità comporta la loro disattivazione.
`Description`	imposta una breve stringa descrittiva usata per identificare l'unità.
`Documentation`	imposta una serie di riferimenti alla documentazione delle operazioni coperte dall'unità.
`Require`	richiede che all'attivazione dell'unità siano attivate pure la/e unità indicate ed alla disattivazione di una di esse sia disattivata anche l'unità stessa.
`Wants`	richiede che all'attivazione dell'unità siano attivate pure la/e unità indicate, ma non comporta la disattivazione dell'unità stessa se viene disattivata una di quelle indicate.

Tabella 5.32: Principali direttive della sezione `[Unit]`.

Oltre alle varie forme di dipendenza appena illustrate, di cui esistono diverse varianti (ad esempio con `Conflicts` si può indicare una dipendenza negativa da altre unità), due direttive usate comunemente sono `Description` che consente di indicare una breve stringa descrittiva usata per identificare l'unità e `Documentation` per inserire riferimenti alla sua documentazione. Si sono riportate le direttive più significative in tab. 5.32, ma ne esistono molte altre (ad esempio la serie delle varie `Condition`*`Qualcosa`* che richiedono la verifica di una serie di condizioni preliminari all'attivazione di una unità) per cui per la documentazione completa si rimanda al solito alla pagina di manuale citata.

La sezione `[Install]` di un *unit file* non viene mai utilizzata nell'esecuzione di `systemd` e serve invece a mantenere informazioni sull'installazione, abilitazione e disabilitazione dell'unità stessa ad uso dei sotto-comandi `enable` e `disable` di `systemctl` (su cui torneremo più avanti). In particolare le direttive `WantedBy` e `RequiredBy`, che prendono come argomento una lista di altre

unità, creano automaticamente nelle rispettive directory `.wants` e `.require` delle stesse i link simbolici all'unità in questione, ottenendo di aggiungerla alle loro dipendenze.

Direttiva	**Significato**
`Alias`	indica una lista di nomi alternativi per l'unità, creati o cancellati come altrettanti link simbolici.
`Also`	richiede di abilitare o disabilitare anche la/e unità indicate quando l'unità viene abilitata/disabilitata.
`RequiredBy`	richiede di inserire l'unità come *richiesta* da quella/e indicate.
`WantedBy`	richiede di inserire l'unità come *voluta* da quella/e indicate.

Tabella 5.33: Principali direttive della sezione `[Install]`.

Un'altra direttiva utilizzata in questa sezione è `Alias` che consente di indicare dei nomi alternativi per l'unità, per cui alla abilitazione dell'unità verranno creati automaticamente altrettanti con quei nomi altrettanti link simbolici all'*unit file* in questione. Le altre principali direttive sono illustrate in tab. 5.33 e di nuovo per la documentazione completa si rimanda alla pagina di manuale citata in precedenza.

Sono inoltre supportate anche delle unità *template*, contraddistinte dal nome che deve terminare con il carattere "`@`". Queste sono molto utili per creare degli *unit file* generici che contengano una configurazioni comuni da cui è possibile creare delle istanze specifiche in fase di esecuzione, in cui i parametri delle direttive vengono referenziati tramite degli opportuni "*specificatori*" avvalorati in fase di attivazione. Ad esempio per gestire i terminali virtuali viene usata l'unità `getty@.service` in modo che quando viene richiesta una console sul primo terminale virtuale possa essere possibile creare automaticamente l'istanza specifica `getty@tty1.service`. In tal caso nel template si farà riferimento all'istanza specifica con lo specificatore " `%I`" che viene automaticamente sostituito dal nome dell'istanza corrente (la stringa fra la "`@`" ed il "`.`", nel caso dell'esempio precedente `tty1`).

Una volta che il sistema è partito `systemd` continua a supportare per compatibilità anche le interfacce del sistema di avvio alla *SysV*; in particolare continua a supportare l'uso del socket `/dev/initctl`, su cui gli si possono inviare i comandi in maniera analoga a quanto si fa con l'`init` tradizionale, e se viene invocato come `init` sulla riga di comando si limita ad eseguire `telinit` con la sintassi vista in sez. 5.3.5. A sua volta con una installazione completa di `systemd` il comando `telinit` non è altro che un link simbolico a `systemctl`, che è il programma di controllo nativo, che consente di gestire tutti gli aspetti dell'avvio del sistema e non solo quelli relativi al cambiamento di *runlevel*, ma che invocato in questo modo si comporta come il `telinit` tradizionale.

Quando viene invocato esplicitamente come `systemctl` il programma prevede che si indichi come primo argomento un comando (i principali sono elencati in tab. 5.34) seguito da eventuali parametri per lo stesso negli argomenti successivi. Se lanciato senza argomenti il programma stampa un elenco di tutte le unità riconosciute, ed è equivalente all'uso del comando `list-units`. Per tutti i comandi che prevedono come argomenti delle unità queste possono essere espresse per nome e non è necessario indicarne il tipo se questo è desumibile dal comando, inoltre se le unità possono essere più di una si possono usare anche dei pattern, usando la sintassi del *filename globbing* della shell.

I comandi più usati sono `start`, `stop` e `restart` che, come con gli script di avvio, consentono di attivare, disattivare o riattivare delle unità (e quindi anche di operare sui servizi). Analogamente si può richiedere di ricaricare i file di configurazione di un servizio con il comando `reload`, ma si

Comando	Significato
`enable`	abilita una o più unità, indicate con i successivi argomenti, creando gli eventuali link simbolici dettati dalla sezione `[Install]`, equivalente dei vari `chkconfig`, `update-rc.d`, ecc.
`disable`	disabilita una o più unità, rimuovendo gli eventuali link simbolici dettati dalla sezione `[Install]`.
`isolate`	attiva l'unità indicata nell'argomento successivo, e tutte le sue dipendenze, e disattiva tutte le altre, l'unità deve essere abilitata con la direttiva `AllowIsolate`.
`mask`	*maschera* una o più unità, indicate con i successivi argomenti, rendendole non attivabili neanche con l'uso esplicito di `start`.
`reload`	richiede la rilettura della configurazione dei servizi delle unità indicate con i successivi argomenti.
`restart`	riattiva (o attiva se non lo era) una o più unità, indicate con i successivi argomenti.
`start`	attiva una o più unità, indicate con i successivi argomenti.
`stop`	disattiva una o più unità, indicate con i successivi argomenti.
`unmask`	rimuove il mascheramento di una unità (annulla un precedente `mask`).
`status`	mostra lo stato di una unità, leggendo anche le eventuali registrazioni relative alla stessa.
`list-units`	elenca le unità presenti, limitandosi a quelle eventualmente indicate negli argomenti successivi.
`daemon-reload`	rilegge la configurazione (e ricarica tutti gli *unit file*).

Tabella 5.34: Principali comandi di `systemctl`.

tenga presente che questo comando, riproducendo l'analogo degli script di avvio, opera solo per la configurazione del servizio stesso. Questo significa che se si modifica un *unit file* la rilettura dello stesso (che è poi la rilettura della configurazione di `systemd`) non viene eseguita da `reload`, e si deve invece usare `daemon-reload`.

Sempre in analogia con gli script di avvio con il comando `status` si può verificare lo stato di una unità, in questo caso però si possono ottenere informazioni molto dettagliate relative a tutti processi ad essa collegati e, appoggiandosi al nuovo sistema di registrazione di `journald`, anche le ultime righe delle registrazioni effettuate nei log dagli stessi.

Oltre alle classiche operazioni disponibili con gli script di avvio, `systemctl` viene usato anche per controllare l'attivazione all'avvio di una o più unità con i comandi `enable` e `disable`, che sono sostanzialmente gli equivalenti di quello che con il sistema tradizionale alla *SysV* veniva fatto con programmi distinti come `chkconfig` o `update-rc.d`.

Dato che con `systemd` non esistono i *runlevel* e l'avvio viene gestito tramite le opportune unità `target`, per richiedere che una unità venga attivata la modalità più comune è quella di inserirle fra quelle "*volute*" dal `target` che si sta usando. Questo è quello che normalmente viene fatto con il comando `enable`, che fa si che vengano creati i link simbolici indicati nella sezione `[Install]` nelle opportune directory `.wanted` o `.required` (a seconda di quanto specificato) e poi ricarica la configurazione di `systemd`. L'operazione inversa viene eseguita da `disable`, che rimuove i suddetti link e disabilita quindi l'attivazione all'avvio. Ovviamente per ottenere che tutto questo abbia un risultato analogo a quello di `chkconfig` o `update-rc.d` è necessario che l'*unit file* sia opportunamente configurato.

Si tenga presente che questi due comandi sono del tutto ortogonali rispetto ai precedenti

`start` e `stop`, abilitare una unità non significa attivarla e disabilitarla non significa disattivarla. Si può attivare anche una unità non abilitata o disattivare una unità abilitata esattamente come si può lanciare un servizio con uno script di avvio senza che questo sia stato inserito sul proprio *runlevel* o fermarlo senza toglierlo dal *runlevel*.

Oltre ad abilitare o disabilitare una unità con `systemctl` si può anche impedire totalmente di attivare una unità, anche rispetto all'uso esplicito di `start`, utilizzando il comando `mask` che ne effettua il cosiddetto "*mascheramento*". Sfruttando il comportamento di `systemd` già citato in precedenza, il comando semplicemente crea sotto `/etc/systemd/system` un link simbolico ad `/dev/null` con il nome dell'unità indicata. Si tenga presente però che se si sono installati direttamente degli *unit file* in detta directory questi non saranno mascherabili (non potendo creare il link simbolico al posto di un file esistente, il comando fallirà). Si può anche mascherare una unità fino al riavvio successivo specificando l'opzione `--runtime` (che crea il link simbolico sotto `/run/systemd/system/`). L'operazione può essere annullata con il comando `unmask`, che semplicemente cancella i link simbolici.

Altri comandi interessanti forniti da `systemctl` sono `kill`, che consente di terminare tutti i processi collegati una unità (che essendo, come accennato, mantenuti all'interno dello stesso *control group*, possono essere facilmente identificati) o `isolate` che consente di simulare il cambiamento di *runlevel*, attivando esplicitamente una unità e tutte le sue dipendenze e fermando tutte le altre. Oltre a questi sono supportati molti altri comandi, e per la documentazione completa si rimanda alla pagina di manuale.

Come accennato in sez. 3.2.3 una delle funzionalità più controverse di `systemd` è quella di aver realizzato un meccanismo indipendente di registrazione degli eventi di sistema, realizzato attraverso il demone ausiliario `journald`. Il comportamento di questo demone viene controllato dal file `/etc/systemd/journald.conf`, che di nuovo segue la sintassi dei *keyfile*, e che prevede una unica sezione `[Journal]`.

La configurazione di default prevede (tramite la direttiva `ForwardToSyslog` preimpostata a `yes`) il reinoltro dei messaggi ad un demone di *syslog* e normalmente non c'è necessità di modificarla. Al più si potrà cambiare la dimensione massima che possono occupare i dati registrati (il demone esegue automaticamente la cancellazione dei dati più vecchi) da effettuare con la direttiva `SystemMaxUse` (il default è il 10% del filesystem). Per maggiori dettagli si rimanda alla pagina di manuale, accessibile con `man journald.conf`.

Il sistema prevede, oltre alla registrazione dei messaggi inviati al *syslog* dai programmi, anche la registrazione degli eventi generati da `systemd` ed oltre alle informazioni dei messaggi di log standard registra pure una serie di ulteriori informazioni (come utenti, PID dei processi, contesti di sicurezza, ecc.), accentrando tutte le informazioni.

Dato che gli eventi vengono registrati, indicizzati e salvati usando un formato binario, essi possono essere riletti solo tramite un programma apposito, `journalctl`, che consente di effettuare le opportune interrogazioni. Se lanciato senza argomenti il comando stamperà tutto il contenuto disponibile nel "*giornale*", partendo dalle voci più vecchie, visualizzato tramite `less`. Dato che in genere si è interessati agli eventi più recenti è possibile richiedere al comando di portarsi automaticamente sulla coda dei dati se si aggiunge l'opzione `-e`.

Lo si può anche usare in modalità analoga al `tail` su un file di log utilizzando l'opzione `-f`, nel qual caso il comando resterà attivo sul terminale continuando a stampare le nuove voci. Si può invece limitare la stampa dei dati dall'ultimo (o n-simo) riavvio usando l'opzione `-b` seguita dal numero dello stesso: se positivo viene contato dall'inizio dei dati, se negativo dalla

Opzione	Significato
-b *N*	limita i risultati a partire dal riavvio indicato da *N*, che può essere un numero positivo (per contare i riavvio dall'inizio dei dati disponibili), negativo (per contare i riavvi dal momento corrente) o un identificativo (valore numerico a 32 bit usato da journald).
-e	si porta automaticamente alla fine dell'elenco delle voci.
-f	resta agganciato al terminale continuando a stampare le nuove voci registrate da journald.
-k	stampa solo i messaggi del kernel.
-p *prio*	stampa solo i messaggi a partire dalla priorità di *syslog* indicata con un valore numerico, partendo da 0 per emerg fino a 7 per debug).
-u *unit*	mostra i messaggi relativi alle unità indicate (come unità singola o come pattern secondo le regole del *filename globbing*).
--since=*data*	stampa solo i messaggi a partire dall'instante indicato (da indicare con il formato di date).

Tabella 5.35: Opzioni per il comando journalctl.

fine, quindi per i dati dall'ultimo riavvio occorre usare journalctl -b -1. Un elenco delle altre opzioni principali è riportato in tab. 5.35, per tutti i dettagli si rimanda al solito alla pagina di manuale.

Nell'uso generico il programma prevede che si passino come argomenti dei filtri di ricerca nella forma *CAMPO=valore*, dove il *CAMPO* indica su quale fra i vari dati registrati da journald si vuole eseguire la ricerca. Ad esempio per ottenere tutti i dati registrati relativi ad una certa unità si dovrà indicare _SYSTEMD_UNIT=*nome.tipo* mentre per ottenere i dati relativi ad un utente si dovrà indicare _UID=*uid*.

Dato che la registrazione dei dati viene effettuata direttamente da systemd alcuni di questi sono considerati "*fidati*" in quanto vengono registrati in maniera diretta e non possono essere modificati da programmi esterni. Per convenzione i campi relativi a questi dati sono identificati, come nei due casi appena citati, da un nome che inizia con un "_".

Campo	Significato
PRIORITY	priorità sul *syslog*, indicata con un valore numerico, partendo da 0 per emerg fino a 7 per debug.
SYSLOG_FACILITY	la *facility* del *syslog*.
SYSLOG_IDENTIFIER	nome identificativo passato al *syslog* (corrispondente in genere al nome del programma).
_CMDLINE	riga di comando con cui è stato lanciato il processo che ha generato la voce.
_COMM	nome del programma eseguito dal processo che ha generato la voce.
_EXE	*pathname* dell'eseguibile del processo che ha generato la voce.
_SYSTEMD_UNIT	unità di systemd che ha generato la voce.
_UID	UID del processo che ha generato la voce.
_PID	PID del processo che ha generato la voce.

Tabella 5.36: Campi di ricerca per journalctl.

In tab. 5.36 si sono riportati alcuni fra i campi di ricerca più rilevanti (per l'elenco completo si rimanda alla pagina di manuale accessibile con man systemd.journal-fields), ma alcune delle ricerche più comuni sono riassunte con una opzione (ad esempio -u e -p). La sintassi del pro-

gramma prevede che se indicano più filtri di ricerca vengano mostrate solo le voci corrispondenti a tutti, ma si può chiedere la corrispondenza ad uno qualunque separandoli con un "+".

Infine si tenga presente che il programma interpreta in maniera intelligente i parametri, per cui se si passa come argomento un pathname questo viene automaticamente trasformato in una ricerca sul campo `_EXE` se corrisponde ad un eseguibile o sul campo `_KERNEL_DEVICE` se corrisponde ad un file di dispositivo.

5.3.7 Gestione generica di servizi, riavvio, spegnimento

Dato che, come appena illustrato, molte distribuzioni stanno passando ad altri sistemi di avvio, l'uso dei classici script di shell sotto `/etc/init.d/` per fermare, avviare o riavviare un servizio potrebbe non essere più possibile. Per consentire di effettuare questa operazione in maniera generica, qualunque sia il sistema di avvio scelto, è allora disponibile il comando `service`, che si incarica di compierla utilizzando le modalità opportune a seconda di quello che si è installato.

Nel caso di avvio alla *System V* il comando si limiterà semplicemente ad invocare lo script di avvio il cui nome sotto `/etc/init.d/` corrisponde al primo argomento, passandogli come argomento il secondo. Se il sistema di avvio è un altro il primo argomento indicherà comunque il nome del servizio su cui agire, ed il secondo argomento l'operazione da eseguire, da specificare usando uno dei valori di tab. 5.29, e sarà cura del comando ottenere la corrispondente azione per il servizio indicato; nel caso di `systemd` ad esempio il primo argomento sarà considerato il nome di una unità di tipo `service` per cui sarà eseguito `systemctl` passandogli come comando il secondo argomento.

Benché tradizionalmente con l'avvio alla *System V* si possa arrestare una macchina o riavviarla usando i *runlevel* 0 e 6, è comunque invalso l'uso di effettuare queste operazioni rispettivamente con i comandi dedicati `halt` e `reboot`. A questi si aggiunge, come sostanziale sinonimo di `halt`, `poweroff`, che richiede esplicitamente anche lo spegnimento fisico della macchina e non solo lo stop di tutti i servizi, ma perché questo avvenga è necessario che sia disponibile l'opportuno supporto nel kernel.

Di nuovo questi comandi formano una interfaccia generica che fornisce la stessa sintassi in maniera indipendente dal sistema di avvio, la cui effettiva realizzazione però dipende da quest'ultimo. Nel caso dell'avvio alla *System V* tutti questi comandi corrispondono sempre allo stesso programma, `halt`, che è quello che effettua la richiesta di arresto, riavvio o spegnimento al kernel. Il comportamento di default prevede che venga preventivamente invocato l'ulteriore comando `shutdown` per arrestare i servizi prima di fermare (o riavviare) definitivamente la macchina. I tre comandi supportano l'opzione `-f` per disabilitare questo comportamento, causando l'arresto o il riavvio immediato senza passare dall'arresto dei servizi. Questo serve perché in alcuni casi l'operazione di arresto dei servizi può fallire o non essere necessaria o voluta.

I tre comandi e questa opzione sono disponibile anche per la versione usata da `systemd` ed ha lo stesso effetto di forzare uno spegnimento o un riavvio immediato senza cercare di fermare i servizi, anche se questo viene effettuato in maniera completamente diversa; in sostanza i tre comandi sono link a `systemctl` che quando invocato con uno di quei nomi esegue il suo corrispondente comando interno. Per le altre opzioni (comuni per tutti e tre i comandi) e per tutti i dettagli si consulti al solito la pagina di manuale.

Benché sia possibile spegnere o riavviare immediatamente la macchina con i comandi appena esaminati, in genere è sempre opportuno eseguire l'operazione con un minimo di preparazione ed

avvisare gli utenti eventualmente collegati, ed è per questo motivo che in questo caso è preferibile fare ricorso al comando `shutdown`, che consente di pianificare un'operazione di fermo macchina, e di notificare gli utenti della stessa.

Il comando prende come primo argomento l'orario a cui deve essere eseguita l'operazione, e come argomento successivo un messaggio da inviare a tutti gli utenti collegati. L'orario può essere indicato in vari formati, a partire da un tempo assoluto specificato nella forma *hh:mm* con ore e minuti, ad un tempo relativo nella forma +*m* dando un numero di minuti; l'esecuzione immediata si ottiene utilizzando `now`, sinonimo di `+0`.

Di nuovo il comando è generico e viene fornito da tutti i sistemi di avvio; nel caso di avvio alla *System V* il default è di portare il sistema in *single user mode* (cioè di porsi sul *runlevel* 1), ma si può richiedere l'arresto della macchina (*runlevel* 0) con l'opzione `-h` o il riavvio (*runlevel* 6) con l'opzione `-r`. Queste due opzioni sono generiche ed hanno lo stesso significato anche con la versione usata da `systemd`, dove di nuovo vengono realizzate tramite `systemctl`. Un'altra opzione generica è `-c` che permette di interrompere uno *shutdown* programmato che ancora non è stato iniziato.

Una volta eseguito, il comando notificherà tutti gli utenti connessi ad un terminale dell'imminenza delle operazioni di arresto, comunicando l'orario programmato, così che questi possano salvare eventuali cambiamenti ai file e scollegarsi, verrà inoltre automaticamente disabilitata la possibilità di collegarsi. La notifica agli utenti viene normalmente fatta con l'ausilio del comando `wall`, che consente di scrivere un messaggio, passato come argomento, su tutti i terminali a cui è collegato un utente, e che può essere usato anche direttamente per inviare notifiche agli utenti collegati, slegate dalla presenza di uno *shutdown*; per i dettagli si consulti al solito la pagina di manuale, con `man wall`.

Opzione	**Significato**
`-c`	annulla uno *shutdown* in corso.
`-h`	esegue lo spegnimento della macchina.
`-k`	non esegue l'operazione ma invia soltanto il relativo messaggio agli utenti.
`-r`	esegue il riavvio della macchina.
`-F`	richiede che al riavvio venga eseguito forzatamente un controllo dei filesystem.
`-f`	richiede che al riavvio non venga eseguito il controllo dei filesystem.
`-t`	specifica il numero di secondi da attendere fra l'invio del segnale `SIGINT` e quello del segnale `SIGKILL` ai processi residui (come per `telinit`).

Tabella 5.37: Principali opzioni del comando `shutdown`

Allo scadere del tempo specificato come argomento verrà effettuata l'operazione richiesta dal comando. Si sono riportate le altre principali opzioni di `shutdown` in tab. 5.37, dove nella prima parte si sono elencate le opzioni supportate in modo generale su tutti i sistemi di avvio, e nella seconda quelle specifiche della versione classica legata all'avvio in stile *System V*. Per l'elenco completo e tutti i dettagli della propria versione si faccia al solito riferimento alla pagina di manuale.

Si tenga presente infine che quando è previsto uno spegnimento o una manutenzione che renderà impossibile l'accesso al sistema, è buona norma avvisare preventivamente gli utenti.

Abbiamo visto che `shutdown` è in grado di farlo al momento della sua invocazione, ma questo vale soltanto per gli utenti che in quel momento sono effettivamente collegati alla macchina. Se si conoscono le necessità con sufficiente anticipo è buona norma segnalare i futuri momenti di fermo a chiunque si collegherà alla macchina tramite i messaggi di benvenuto che vengono stampati all'accesso, controllati dai file visti in sez. 4.3.5. Sarà cioè opportuno inserire un opportuno avviso o in `/etc/motd` o all'interno di `/etc/issue` e `/etc/issue.net`.

5.4 La gestione di interfacce e periferiche

Tratteremo in questa sezione gli argomenti relativi all'utilizzo di una serie di tipiche interfacce hardware di cui sono dotati i moderni computer, e come, con le versioni più recenti di Linux, il relativo supporto venga gestito in maniera il più possibile automatica attraverso sistemi come *hotplug* e *udev*.

5.4.1 Gestione delle interfacce di espansione

Una delle caratteristiche essenziali dell'architettura hardware dei computer moderni è quella di disporre di opportune interfacce di espansione in cui inserire delle schede dedicate che permettono l'uso di nuove periferiche; si ha così la possibilità di ampliare le capacità di un computer utilizzando degli appositi dispositivi in grado effettuare i più svariati compiti.

Una interfaccia di espansione in sostanza non fornisce direttamente delle funzionalità finali, che sono specifiche del singolo dispositivo che si pone su di essa, quanto una interfaccia hardware comune che consente una comunicazione fra la CPU e la memoria e tutti i dispositivi che sono posti su detta espansione. In questo modo il sistema operativo può inviare comandi e scambiare dati con le singole periferiche, in modo da poterne utilizzare le capacità specifiche.

In Linux il supporto per queste interfacce è ovviamente fornito direttamente dal kernel, che predispone tutta l'infrastruttura software con cui è possibile leggere i dati relativi a dette interfacce, e poi dialogare con i singoli dispositivi su di esse presenti. Questi ultimi, seguendo il criterio fondamentale dell'architettura di un sistema unix-like per cui *tutto è un file*, potranno poi essere acceduti attraverso l'interfaccia generica dei file illustrata in sez. 1.2.1, con l'eccezione delle interfacce di rete, che non rientrando bene in questa astrazione, sono utilizzate in modo diverso.

Le due interfacce di espansione tradizionalmente più utilizzate nel mondo dei computer basati sulla architettura classica dei cosiddetti PC (in sostanza la piattaforma basata su Intel o compatibili) sono la ormai superata ISA (*Industry Standard Architecture*) e la più recente ed ampiamente utilizzata PCI (*Peripheral Component Interconnect*). Entrambe queste interfacce definiscono uno standard sia hardware (consistente in piedinatura, dimensioni, specifiche dei segnali elettrici ecc.) che software (i comandi che la CPU deve dare per comunicare con l'interfaccia ed i dispositivi presenti sulla stessa). Ad oggi ISA è completamente abbandonata, mentre lo standard PCI ha subito diverse evoluzioni, ma i concetti di base restano comunque sostanzialmente gli stessi.

La struttura di base di queste interfacce è quella di definire uno speciale canale di comunicazione (il cosiddetto *bus*) sul quale far passare sia i dati che i comandi che vanno dal *sistema centrale* (CPU e memoria) ai dispositivi inseriti nell'interfaccia stessa (le *periferiche* appunto), e

viceversa. Le interfacce forniscono inoltre un opportuno meccanismo che permetta di comunicare in maniera indipendente (identificando opportunamente i flussi di dati sul *bus*) con ciascuna delle periferiche presenti sull'interfaccia.

Una descrizione dettagliata del funzionamento di queste interfacce va ben oltre lo scopo di questo testo, la loro gestione è infatti completamente realizzata all'interno del kernel attraverso il codice relativo al loro supporto. Quello che interessa dal punto di vista dell'amministrazione di sistema è capire quali sono le risorse utilizzate e come allocarle e le funzionalità messe a disposizione del kernel che consentono di accedere alle informazioni relative a dette interfacce ed ai dispositivi su di esse presenti a scopo di configurazione o di controllo.

Le risorse fondamentali usate da una interfaccia sono sostanzialmente due gli *interrupt* e i *canali DMA*. Un *interrupt* è un meccanismo con cui un dispositivo hardware può inviare un segnale al processore[49] (usando quella che si chiama una *linea di interrupt*) così che questo possa *interrompere* le operazioni e rispondere all'esigenza di attenzione così manifestata dal dispositivo. In genere un processore ha un numero limitato di linee di *interrupt*; originariamente nei PC erano 8 e poi sono state portate a 15, in altre architetture sono 32 o 64.

Nei computer più recenti, con il crescere del numero di dispositivi e la presenza di sistemi multiprocessore la gestione degli *interrupt* è stata demandata ad un apposito sistema, denominato APIC, da *Advanced Programmable Interrupt Controllers*, questo oltre a fornire un numero di *interrupt* molto più elevato, la definizione di priorità e l'allocazione dinamica degli stessi, consente anche di risolvere i problemi connessi alla presenza di più processori (in tal caso si tratta di decidere a quale processore inviare l'*interrupt* perché lo processi) realizzando un meccanismo di smistamento fra gli stessi.

Il sistema prevede la presenza di due componenti, l'APIC locale (LAPIC), presente nella misura di una unità per processore, che gestisce gli *interrupt* destinati al singolo processore e quelli creati fra processori, e l'I/O APIC, presente nella misura di una unità per bus di periferiche, che gestisce gli interrupt ricevuti dalle periferiche stesse e li instrada verso gli APIC locali, per il trattamento da parte dei singoli processori.

La seconda risorsa è quella dei *canali DMA*, questi sono un meccanismo hardware che consente dei trasferimenti diretti di dati da una periferica alla memoria senza che questi debbano essere letti direttamente con l'intervento della CPU, che può essere utilizzata in altre operazioni. Con l'uso dei canali DMA il sistema si limita a richiedere solo un intervento iniziale della CPU per indicare al dispositivo in quale zona di memoria inviare i dati, che saranno poi trasferiti in maniera asincrona. In genere l'uso di un canale DMA si abbina sempre a quello di un *interrupt* che serve a segnalare la conclusione del trasferimento, così che la CPU venga avvertita che è possibile utilizzare i dati disponibili in memoria.

Uno dei problemi relativi alla gestione delle interfacce di espansione è allora proprio quello della allocazione degli *interrupt* e dei canali DMA ai dispositivi esistenti,[50] che di norma può essere eseguita a livello di BIOS. In particolare alcuni *interrupt* sono assegnati staticamente a periferiche presenti sui PC da prima che fosse possibile una allocazione dinamica, come l'interfaccia IDE per i dischi, le seriali e la parallela; gli altri poi possono essere lasciati liberi per l'uso

[49] nei sistemi tradizionali questo viene fatto fisicamente alzando un livello su uno dei piedini del processore a questo dedicato.

[50] in realtà questo è un problema che si aveva quasi esclusivamente sui vecchi PC intel-compatibili, che avevano pochi *interrupt* e canali DMA e tutta una serie di limitazioni ereditate dall'architettura originaria che su altre piattaforme non esistono.

da parte delle interfacce di espansione o di altre interfacce interne come USB (che vedremo in sez. 5.4.4).

I problemi che possono sorgere sono allora quelli dell'allocazione di queste risorse; il kernel permette di esaminare lo stato corrente di queste allocazioni attraverso il file `/proc/interrupts` per gli interrupt e `/proc/dma` per i canali DMA; un esempio potrebbe essere il seguente:

```
# cat /proc/interrupts
           CPU0
  0:     584706          XT-PIC  timer
  1:      15435          XT-PIC  keyboard
  2:          0          XT-PIC  cascade
  8:          4          XT-PIC  rtc
  9:          2          XT-PIC  usb-uhci, usb-uhci, btaudio, bttv
 10:     933383          XT-PIC  EMU10K1, eth0
 11:       2170          XT-PIC  ide2, ide3, aic7xxx
 12:     319229          XT-PIC  PS/2 Mouse
 14:         56          XT-PIC  ide0
 15:     162630          XT-PIC  ide1
NMI:          0
LOC:     584659
ERR:        848
MIS:          0
```

che mostra l'allocazione degli *interrupt*, il cui numero progressivo è riportato in prima colonna, rispettivamente ai dispositivi ad essi associati (riportati nell'ultima); nella seconda colonna sono riportati il numero di *interrupt* registrati al momento ed un sommario di questa statistica è riportato nelle ultime quattro righe. Analogamente abbiamo:

```
# cat /proc/dma
 4: cascade
```

che mostra l'allocazione dei canali DMA.

L'uso di APIC sulle macchine più moderne permette in genere di risolvere i problemi di allocazione degli *interrupt*, ma può accadere che ne introduca a sua volta di altri. In particolar modo a causa di errori sia nell'implementazione dell'hardware, che nel supporto da parte del kernel, ci si può trovare con un sistema non funzionante. Per questo motivo su Linux è disponibile fra le opzioni di avvio del kernel (vedi sez. 5.3.1) `noapic` che disabilita l'uso di questo sistema, o `nosmp` che inoltre disabilita anche l'uso in modalità multiprocessore. Si tratta di soluzioni di emergenza a cui si può provare a ricorrere in caso si manifestino problemi.

Una terza risorsa usata da alcune periferiche è quella delle *porte di I/O*, una modalità di comunicazione diretta fra la CPU e le stesse effettuata attraverso l'accesso ad alcune locazioni di memoria riservate (chiamate appunto in questo modo) leggendo e scrivendo dalle quali si andava a leggere e scrivere direttamente sui dispositivi. Anche in questo caso alcune di queste porte sono allocate staticamente a periferiche standard come le seriali, mentre altre possono dover essere allocate opportunamente quando si inserisce il relativo dispositivo su una interfaccia di espansione; la situazione corrente dell'allocazione è riportata dal kernel nel file `/proc/ioports`, un cui esempio è:

```
# cat /proc/ioports
0000-001f : dma1
0020-003f : pic1
```

```
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
02e8-02fe : serial(set)
02f8-02ff : serial(set)
0376-0376 : ide1
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial(set)
0cf8-0cff : PCI conf1
5c20-5c3f : ALi Corporation M7101 PMU
...
```

Di nuovo con tutto l'hardware recente questa allocazione viene gestita in maniera dinamica e trasparente per l'utente, ci sono casi però, specie con vecchie periferiche, in cui non è così e può essere necessario dover effettuare una allocazione manuale (in genere tramite il BIOS).

Una delle prime interfacce di espansione realizzate nei PC tradizionali è stata la ISA, che nacque come estensione dei bus dei vecchi PC, che consentivano l'utilizzo di schede di espansione dedicate. Originariamente l'interfaccia prevedeva un bus a 8 bit, quasi subito portato a 16. Veniva usata principalmente per modem interni e schede sonore ed al giorno d'oggi è totalmente in disuso, e si trova soltanto su delle macchine molto vecchie.

Le interfacce di espansione più utilizzate nei computer odierni sono quelle che utilizzano lo standard PCI *Peripheral Connect Interface*, che venne creato per fornire un bus di espansione generico che fosse adatto all'evoluzione dei computer, ed in grado di supportare velocità di trasferimento dei dati molto superiori al precedente ed ormai completamente abbandonato ISA. Il bus nasce a 32 bit e prevede una frequenza per le operazioni di 33.3MHz, per una banda passante teorica di circa 1Gbit/s. In seguito sono state realizzate varie estensioni dello standard, fra cui il bus AGP delle schede video (che è sostanzialmente un PCI con frequenze operative più alte), ed ulteriori evoluzioni come il PCI-X ed il PCI Express, che riguardano diversi formati per le schede, uso di bus a 64 bit e maggiori velocità di trasferimento.

Nel caso di PCI la allocazione delle risorse è dinamica; in genere gli *interrupt* vengono associati sulla base dello slot su cui è inserita la scheda di espansione, inoltre il bus supporta la condivisione degli *interrupt*. Un'altra caratteristica specifica del bus PCI è che ogni scheda riporta al suo interno (nel *firmware*) una serie di informazioni fra cui una coppia di numeri che specificano sia il produttore che il tipo della scheda, che possono essere confrontati con un database delle schede prodotte (usualmente mantenuto nel file `/usr/share/misc/pci.ids`).

In un bus PCI ciascuna periferica viene identificata univocamente grazie ad un indirizzo che è composto da tre numeri: il *numero di bus*, il *numero di dispositivo* ed il *numero di funzione*. Il *numero di bus* identifica su quale bus si trova la scheda, infatti lo standard supporta la possibilità di avere più bus (fisicamente distinti) sulla stessa macchina, fino ad un massimo di 256. Una caratteristica comune del PCI è che i bus possono essere collegati fra di loro attraverso dei dispositivi appositi detti *PCI bridge*, per cui con una scheda apposita si può controllare un altro bus PCI contenente altre schede. Per ciascun bus si possono poi inserire fino a 32 schede

diverse, identificate per *numero di dispositivo*, che possono supportare fino a 8 diverse periferiche cadauna (in caso di schede *multifunzione*). Il tutto compone un numero a 16 bit che viene a costituire l'*indirizzo* hardware della periferica sul bus PCI.

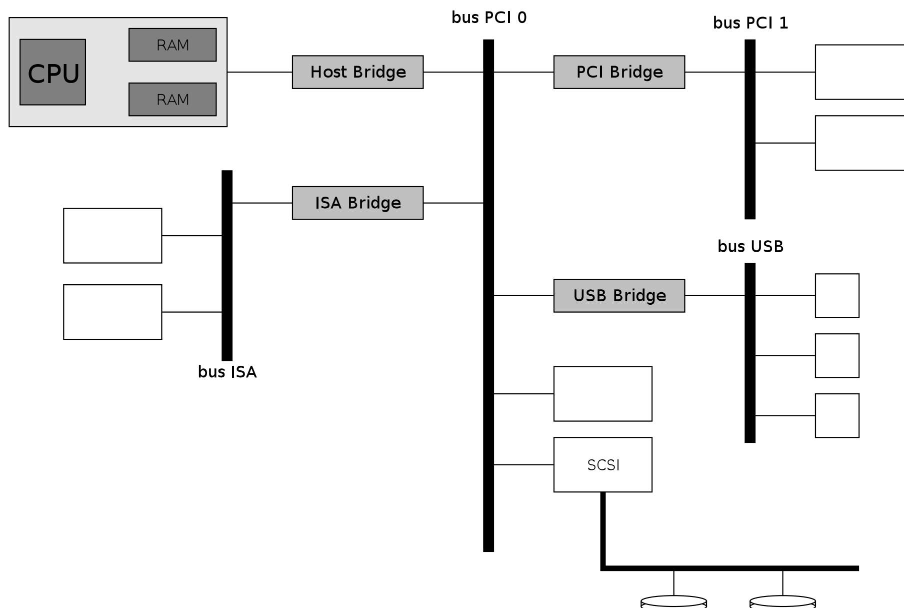


Figura 5.4: Schema della disposizione del bus PCI e delle varie interfacce di espansione.

I computer più recenti in genere hanno sempre almeno due bus, la CPU è collegata attraverso il cosiddetto *host bridge* sul bus principale che è sempre il bus 0, ulteriori bus sono agganciati su questo con un *PCI bridge* e su di essi possono essere agganciati ulteriori bus in modo da formare un albero. Uno schema della disposizione dei bus nelle architetture PC è in fig. 5.4.

In generale il kernel riporta le informazioni relative a tutte le periferiche disponibili sul bus PCI all'interno del filesystem *proc* nella directory `/proc/bus/pci`. All'interno della directory è presente, per ogni bus PCI presente nel sistema, una sottodirectory identificata con il numero assegnato a ciascun bus che contiene un file per ciascuna periferica nella forma `NumeroDispositivo.NumeroFunzione`, ed il file `devices` in cui sono riportate le informazioni relative a tutte le schede presenti (ed agli eventuali moduli cui sono associate).[51]

La lista dei dispositivi presenti sul bus può essere ottenuta anche con il comando `lspci`; nei nuovi kernel questo è il metodo canonico, e sarà l'unico supportato in futuro. Il comando non prende argomenti e stampa sullo standard output la lista delle schede rilevate sul bus PCI, con qualcosa del tipo:

```
[root@gont corso]# lspci
00:00.0 Host bridge: VIA Technologies, Inc. VT8363/8365 [KT133/KM133] (rev 03)
00:01.0 PCI bridge: VIA Technologies, Inc. VT8363/8365 [KT133/KM133 AGP]
```

[51] nelle vecchie versioni del kernel esisteva solo il file `/proc/pci`, contenente una lista descrittiva delle schede presenti, eliminato nella serie 2.6.x, e presente per compatibilità nella serie 2.4.x.

```
00:07.0 ISA bridge: VIA Technologies, Inc. VT82C686 [Apollo Super South] (rev 4
0)
00:07.1 IDE interface: VIA Technologies, Inc. VT82C586/B/686A/B PIPC Bus Master
IDE (rev 06)
00:07.2 USB Controller: VIA Technologies, Inc. USB (rev 16)
00:07.3 USB Controller: VIA Technologies, Inc. USB (rev 16)
00:07.4 Host bridge: VIA Technologies, Inc. VT82C686 [Apollo Super ACPI] (rev 4
0)
00:09.0 SCSI storage controller: Adaptec AHA-2940U/UW/D / AIC-7881U (rev 01)
00:0f.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL-8139/8139C/813
9C+ (rev 10)
01:00.0 VGA compatible controller: nVidia Corporation NV11 [GeForce2 MX/MX 400]
 (rev a1)
```

Il comando riporta nella prima colonna l'indirizzo di ciascun dispositivo ordinato per numero di bus, numero di dispositivo e numero di funzione seguito da una descrizione sommaria della periferica relativa; usando l'opzione `-v` si può avere una descrizione più dettagliata comprensiva delle risorse utilizzate dalla periferica stessa, con qualcosa del tipo di:

```
0000:00:12.0 Ethernet controller: VIA Technologies, Inc. VT6102 [Rhine-II] (rev 74)
        Subsystem: Micro-Star International Co., Ltd.: Unknown device 7120
        Flags: bus master, medium devsel, latency 32, IRQ 23
        I/O ports at dc00 [size=256]
        Memory at dffffe00 (32-bit, non-prefetchable) [size=256]
        Capabilities: <available only to root>
```

e ripetendo una seconda volta l'opzione si possono avere ancora più dettagli.

Con l'opzione `-t` si può invece avere una stampa della struttura ad albero del bus, mentre con `-s` si può selezionare quali dispositivi guardare, passando un parametro che esprime il relativo indirizzo nella forma `bus:slot.func` (il formato della prima colonna del precedente esempio) dove ciascun numero identificativo può essere sostituito dal carattere jolly "$*$". Le altre opzioni principali sono riportate in tab. 5.38, per un elenco completo si faccia al solito riferimento alla pagina di manuale del comando.

Opzione	**Significato**
`-b`	riporta la lista degli *interrupt* per come li vede il bus (con APIC nel kernel vengono rimappati).
`-d`	mostra le informazioni relative ai dispositivi di un singolo produttore (usando gli identificativi della scheda sulla base del valore del parametro `vendorID:deviceID`) dove entrambi gli identificativi sono espressi come numeri esadecimali.
`-i`	usa il file passato come parametro come database degli identificativi delle schede PCI.
`-n`	riporta il valore numerico degli identificatori delle schede invece di usare la descrizione testuale riportata nel database.
`-s`	mostra le informazioni relative ad uno o più dispositivi che corrispondono ad un certo indirizzo sul bus espresso da un parametro nella forma `bus:slot.func`.
`-t`	mostra una schematizzazione ad albero della disposizione dei dispositivi.
`-v`	aumenta le informazioni stampate (può essere ripetuto due volte).

Tabella 5.38: Principali opzioni per il comando `lspci`.

Un secondo comando utilizzabile per operare sul bus PCI è `setpci`, che può essere utilizzato per interrogare e configurare i singoli dispositivi presenti sul bus. Il comando necessita sempre

di una opzione di selezione per indicare su quale dispositivo operare, questa può essere sia `-s` per usare l'indirizzo che `-d` per usare l'identificativo della scheda; entrambe usano la stessa sintassi già vista in tab. 5.38 per `lspci`.

Il comando prende come argomento il nome del registro su cui si vuole operare. Lo standard PCI prevede infatti che tutte le schede debbano avere una serie di registri di configurazione, che contengono informazioni o controllano vari aspetti del loro funzionamento, come il tempo massimo che un dispositivo può tenere il bus.

Indicando solo il nome di un registro ne sarà stampato il valore corrente, indicando una assegnazione nella forma `registro=valore` ne sarà invece cambiato il contenuto (si tenga conto che il valore deve essere espresso in esadecimale). Al nome del registro si possono aggiungere i suffissi `.B`, `.W` e `.L` per indicare che si vuole eseguire l'operazione su un registro di dimensione pari ad un byte, una parola (16 bit) o una parola lunga (32 bit). Al posto del nome può anche essere usato un valore esadecimale che ne indica la posizione nello spazio dei registri.

Un elenco sommario dei principali registri e del relativo significato è riportato in tab. 5.39, un elenco completo di tutti i nomi definiti è riportato nella pagina di manuale di `setpci` (i nomi sono riportati in maiuscolo, ma possono essere specificati anche in minuscolo), per il relativo significato si può fare riferimento alla dichiarazione delle costanti omonime in `/usr/include/linux/pci.h` o alle specifiche dello standard PCI.

Registro	**Significato**
`device_id`	identificativo del dispositivo.
`vendor_id`	identificativo del produttore del dispositivo.
`latency_timer`	imposta un temporizzatore scaduto il quale il dispositivo rilascia l'uso del bus (così si permette un uso più corretto del bus da parte di altri dispositivi presenti).
`min_gnt`	valore (in sola lettura) del tempo minimo per il quale deve essere garantito l'uso del bus al dispositivo (in unità di quarti di microsecondo).
`max_lat`	valore (in sola lettura) che specifica quanto spesso il dispositivo necessita di accedere al bus (in unità di quarti di microsecondo).

Tabella 5.39: Costanti identificative di alcuni registri PCI usate dal comando `setpci`.

5.4.2 Gestione delle interfacce SCSI

L'interfaccia SCSI (*Small Computer System Interface*) potrebbe essere inserita fra le interfacce di espansione generiche trattate in sez. 5.4.1 in quanto anche essa definisce una struttura a *bus* su cui vengono innestati dispositivi multipli. La trattiamo a parte in quanto in realtà questa non è necessariamente una interfaccia collegata ad un solo PC (è infatti possibile, ad esempio con sistemi di dischi condivisi, collegare più PC allo stesso bus SCSI) e di norma viene realizzata tramite l'uso di apposite schede (i cosiddetti *controller SCSI*) che si inseriscono in una delle interfacce precedentemente citate e vengono utilizzati tramite esse.

Una seconda differenza con le interfacce generiche di sez. 5.4.1 è che nonostante sia possibile inserire dispositivi diversi su un bus SCSI, l'interfaccia è utilizzata quasi esclusivamente per l'accesso a periferiche di stoccaggio di dati (principalmente dischi e nastri, ma anche CD e masterizzatori) e non all'uso di periferiche generiche, ed il protocollo stesso della trasmissione dei dati sul bus è stato pensato per questo tipo di lavoro.

Unica eccezione significativa, restata in uso per qualche tempo, è quella dei primi scanner che, non essendo all'epoca disponibile una interfaccia più semplice con sufficienti capacità, venivano pilotati da una interfaccia SCSI. Oggi probabilmente l'unico altro uso "atipico" è quello di controllo per i dispositivi cambia-nastri delle cosiddette *tape factory*, il cui uso comunque è sempre più marginale.

Uno dei problemi maggiori con le interfacce SCSI è che dalla prima definizione dello standard (SCSI-1, del 1986) si sono susseguite molte modifiche: lo SCSI-2, cui segue lo SCSI-3 che però non è mai stato rilasciato come tale ed è stato suddiviso in parti distinte dell'interfaccia. Ma anche all'interno degli stessi standard le modalità di realizzazione delle cablature, dei connettori e dei segnali sono le più varie, il che ha portato ad una discreta confusione.

Lo standard originario (SCSI-1) prevedeva un bus basato su un connettore a 50 pin, di cui 8 (più uno di parità) erano riservati per la trasmissione dei dati, la frequenza di trasmissione era di 5MHz, con una corrispondente velocità massima di trasferimento di 5MB/s, erano poi previste 4 linee per gli indirizzi, consentendo fino ad un massimo di 8 periferiche sul bus.

Una prima modifica, chiamata *Fast SCSI* venne fatta aumentando la frequenza del bus a 10MHz per raddoppiare la velocità di trasferimento. Altre modifiche vennero fatte riguardo la modalità di gestione dei segnali sui cavi ed il tipo di cablatura, questo portò allo sviluppo di un secondo standard (lo SCSI-2). Questa seconda versione prevedeva una lunga serie di estensioni, molte delle quali relative alla gestione dei segnali sui cavi ed alla composizione della cablatura stessa (terminazione attiva, segnali differenziali, nuovi connettori), oltre ad un aumento dei comandi disponibili e all'introduzione del sistema del *command queing* che permetteva ad un singolo dispositivo di accettare più comandi, in modo da poterne ottimizzare l'ordine di esecuzione.

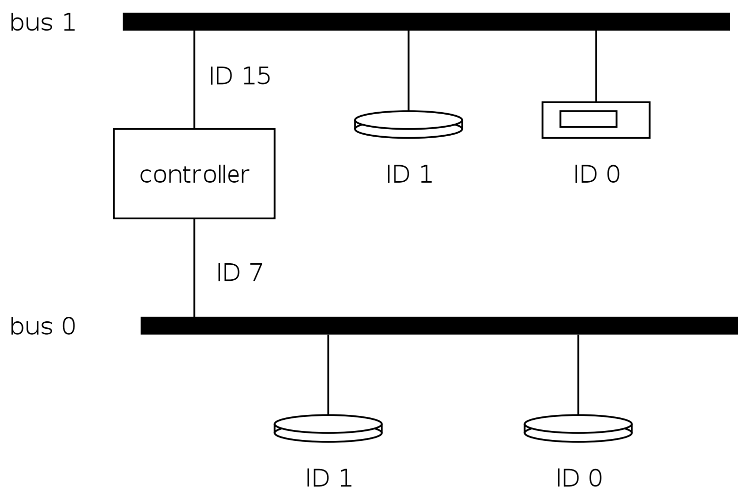


Figura 5.5: Schema della struttura delle interfacce SCSI.

Una delle modifiche principali (detta *Wide SCSI*), che richiede un connettore diverso a 68 pin, prevedeva la possibilità di usare un bus per il trasferimento dati a 16 bit (detto *wide*, in contrapposizione con il precedente, detto *narrow*) raddoppiando anche il numero di periferiche

inseribili nel bus. In questo modo si otteneva a parità di frequenza (con il cosiddetto *Fast Wide SCSI*) un raddoppio della velocità di trasferimento 20MB/s.

Passi successivi sono state l'introduzione di bus con frequenze sempre maggiori consentendo velocità di trasferimento sempre maggiori: *Ultra SCSI* a 40MB/s, *Ultra2* a 80MB/s, *Ultra160* a 160MB/s e ultimamente pure *Ultra 320* a 320MB/s e *Ultra 640* a 640MB/s. Inoltre è stato introdotto pure il SAS, *Serial Attached SCSI* in cui si è abbandonato il bus parallelo per passare ad un collegamento di tipo seriale ad alta velocità mantenendo il protocollo dei comandi SCSI, come avvenuto con il *Serial ATA*. Infine con iSCSI ci si è resi indipendenti dal mezzo di collegamento fisico, facendo passare i comandi del protocollo sulla rete tramite schede Ethernet.

Lo schema classico di una interfaccia SCSI è riportato in fig. 5.5, il bus prevede l'esistenza di almeno una *unità di controllo*, la scheda, detta *controller SCSI*, che in genere si mette sul bus PCI del computer, e a cui si collegano le altre periferiche utilizzando gli appositi connettori. Un singolo controller può gestire anche più bus, nel qual caso sarà in grado di alloggiare più connettori. Tutte le periferiche sul bus, compresa l'unità di controllo, vengono identificate attraverso un indirizzo di identificazione, detto SCSI ID.

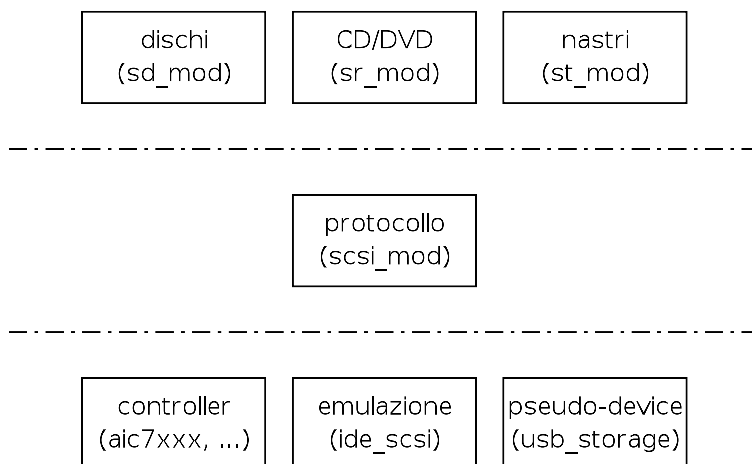


Figura 5.6: Strutturazione del supporto alle interfacce SCSI nel kernel.

Nel caso di Linux il supporto SCSI è presente fin dalle prime versioni del kernel, ma dato che lo standard prevede vari tipi di periferiche a partire dal kernel 2.4 il sistema è stato organizzato su tre livelli, secondo lo schema illustrato in fig. 5.6. Al livello più alto, utilizzati direttamente dai programmi quando accedono ai relativi file di dispositivo, stanno i driver per le periferiche che si trovano sul bus, come CD, nastri o dischi, ciascuno dei quali richiede una modalità d'accesso distinta; a questi si aggiunge il supporto generico che viene usato come interfaccia per inviare direttamente comandi ad una periferica (che viene usato per pilotare dispositivi particolari come gli scanner o i masterizzatori).

Il livello intermedio costituisce il collante fra i driver dei dispositivi finali ed i driver che invece si occupano di gestire i controller per l'accesso al bus, che saranno diversi a seconda della scheda SCSI utilizzata. Oltre a quest'ultimi però si situano a questo livello anche tutti i moduli che consentono di utilizzare il protocollo SCSI per controllare dispositivi posti su altre interfacce. Ad

esempio il protocollo viene utilizzato per accedere ai dischi sia sul bus USB (che siano memorie a stato solido o veri e propri hard disk) che *Serial ATA*, sia per utilizzare i masterizzatori IDE attraverso un meccanismo di emulazione. In tal caso i comandi del protocollo SCSI, invece di diventare segnali elettrici su connettore attaccato ad un controller, saranno inviati su un bus virtuale fornito dal driver per il supporto di dette funzionalità.

Le periferiche accedute tramite interfaccia SCSI sono identificate attraverso quattro numeri: il primo è l'*host adapter number* che identifica, come il nome stesso indica, quale è l'interfaccia (in genere la scheda col controller, ma può essere anche l'interfaccia di un bus virtuale) con cui si accede ai dispositivi; il numero viene assegnato dal kernel all'avvio a seconda dell'ordine in cui rileva le schede presenti o di quando viene abilitato il supporto per bus virtuali che si utilizzeranno. L'*host adapter number* è un numero progressivo che parte da zero.

Il secondo numero è il cosiddetto *channel number*, che identifica ciascun bus (detto anche, in altra nomenclatura, *canale SCSI*) associato ad un *host adapter*; alcuni controller infatti possono gestire più di un bus, e questi saranno numerati progressivamente a partire da zero, nell'ordine in cui l'adattatore stesso li presenta (che dipenderà ovviamente da come è fatto quest'ultimo). Anche questo è un numero progressivo che parte da zero.

All'interno di ciascun *canale* si avrà poi l'identificativo del singolo dispositivo posto su di esso (lo SCSI ID di cui abbiamo già parlato). In genere sui dispositivi questo viene stabilito dall'utente con degli appositi interruttori; di norma sono presenti dei jumper che permettono di impostare le linee corrispondenti, anche se in alcuni confezionamenti sono disponibili dei selettori. Per i controller invece l'identificativo può essere modificato dal BIOS di configurazione, ma è in genere preimpostato al valore più alto (in genere 7) che è quello che ha maggiore priorità.

Infine alcuni dispositivi più sofisticati (come le unità a nastro dotate di libreria, o i juke-box di CDROM) possono avere al loro interno diverse funzionalità che vengono allora indirizzate da un quarto numero detto *Logical Unit Name*, o LUN. In questo caso la periferica da usare (ad esempio l'unità a nastro o il meccanismo di controllo della libreria) viene identificata completamente usando anche il LUN; per la maggioranza dei dispositivi, che non hanno più periferiche a bordo, questo è nullo. Pertanto una modalità di indicare le periferiche SCSI è quella di fornire una quadrupletta di numeri che indicano interfaccia, canale, ID e LUN, del tipo di:

```
<scsi(_adapter_number), channel, id, lun>
```

e quando esse vengono riconosciute dal kernel in fase di avvio si otterranno nei log (o nell'uscita di `dmesg`) dei messaggi del tipo:

```
dmesg
scsi0 : SCSI emulation for USB Mass Storage devices
  Vendor: IC35L120  Model: AVV207-0          Rev:  0 0
  Type:   Direct-Access                      ANSI SCSI revision: 02
USB Mass Storage device found at 2
...
SCSI device sda: 241254720 512-byte hdwr sectors (123522 MB)
sda: assuming drive cache: write through
 sda: sda1
Attached scsi disk sda at scsi0, channel 0, id 0, lun 0
```

e si noti come in questo caso sia stato rilevato un disco su USB, in cui l'interfaccia di emulazione viene vista come `scsi0`, ed il dispositivo viene riconosciuto (non essendo qui disponibile un

meccanismo di assegnazione degli ID ogni dispositivo verrà associato ad una diversa interfaccia) con valori del canale, ID e LUN nulli.

Con il kernel 2.6 l'informazione relativa ai dispositivi SCSI è stata rimossa dalla directory `/proc/scsi/` e viene mantenuta sotto `/sys` grazie al filesystem *sysfs*. Questo però la rende assai meno fruibile e per questo è disponibile un comando specifico, `lsscsi` che consente di ottenere un elenco di tutti i dispositivi SCSI presenti nel sistema. Eseguendolo si otterrà qualcosa del tipo:

```
root@selinor:~# lsscsi
[0:0:0:0]    disk    FUJITSU  ETERNUS_DXL      0000  /dev/sdb
[0:0:0:1]    disk    FUJITSU  ETERNUS_DXL      0000  /dev/sdc
[1:0:0:0]    disk    ATA      ST3250310AS      4.AA  /dev/sda
[5:0:0:0]    cd/dvd  TSSTcorp DVD-ROM TS-H353C FS02  /dev/sr0
```

dove in prima colonna viene riportata la quadrupletta di numeri che costituiscono l'identificativo del dispositivo sul bus SCSI, seguita dalla sua tipologia (disco, CD o altro), e da alcuni dati relativi al produttore ottenuti dal bus stesso. Infine in ultima colonna viene riportato il file di dispositivo con cui vi si può accedere.

Rispetto agli equivalenti dispositivi IDE classici in genere dischi e CDROM SCSI hanno prestazioni superiori per la capacità del bus di gestire l'invio di comandi multipli, vista però la maggiore complessità del protocollo sono anche in genere più costosi, per questo CD e masterizzatori sono ormai praticamente inesistenti da anni, e rimangono sostanzialmente solo dischi e nastri per l'uso professionale, anche se i primi subiscono sempre di più la concorrenza da parte del *Serial ATA* che ha prestazioni sostanzialmente equivalenti e che comunque viene visto con la stessa interfaccia.

Un secondo aspetto da tenere presente nel caso di dispositivi SCSI classici è che in genere il cablaggio degli stessi è piuttosto delicato, alcuni bus infatti devono essere opportunamente *terminati* per consentire una corretta trasmissione dei segnali, inoltre la presenza di diversi standard sullo stesso bus può portare a degrado di prestazioni. Infine occorre tenere sotto controllo l'allocazione degli ID, che è di norma a carico degli utenti, la presenza di due dispositivi con lo stesso ID infatti li rende entrambi inutilizzabili.

Nelle distribuzioni recenti i programmi di gestione dell'interfaccia SCSI sono distribuiti con il pacchetto `scsitools`; il principale è `scsiinfo` che permette di interrogare un dispositivo (passato come argomento) per ricavarne tutta una serie di proprietà (per l'elenco completo e le relative spiegazioni si consulti la pagina di manuale), è invece da segnalare l'uso dell'opzione `-l`, da usare senza specificare un dispositivo, che riporta quelli presenti.

5.4.3 Gestione delle interfacce seriali

Le interfacce seriali sono una delle prime interfacce create per la comunicazione fra computer, e prevedono appunto una comunicazione via cavo estremamente semplice (e di breve distanza) basata sull'invio dei dati lungo una linea di trasmissione, in cui un dato viene trasmesso appunto come una sequenza di singoli segnali che traducono direttamente il valore dello stesso espresso come sequenza di bit.

Benché oggi stiano sparendo, le seriali sono una delle interfacce di comunicazione più elementari, presenti fin dai primi PC. Fino a qualche tempo fa tutti i PC montavano almeno due interfacce, ma oggi, data la lentezza della comunicazione e la presenza di nuove interfacce più

veloci (come l'USB) sono praticamente scomparse, anche se resta la disponibilità di adattatori USB che consentono comunque di utilizzare questo tipo di interfaccia. Ciò non di meno esse restano le interfacce tipiche con cui vengono gestiti i modem telefonici,[52] ed per la loro semplicità realizzativa una delle interfacce più comuni usate per collegarsi ad apparati esterni come router, switch programmabili o dispositivi industriali.

A causa delle limitazioni dell'architettura originale dei PC, l'uso contemporaneo di due porte seriali crea problemi se queste condividono la stessa linea di *interrupt*. Per questo quando era necessario un numero maggiore di porte seriali occorreva utilizzare apposite schede di estensione. Esistono ad esempio delle schede dedicate ad alta velocità, usate per lo più dai provider per gestire i modem, per le quali il kernel è in grado di supportare la condivisione degli *interrupt*. Un altro caso di conflitto è quello in cui, come parte di una scheda ISDN o di un modem-fax interno, diventa disponibile una nuova interfaccia seriale e bisogna verificare che questa non sia sulla stessa linea di *interrupt* di una porta seriale che è già in uso.

Tutte queste problematiche oggi sono sostanzialmente scomparse, come le schede seriali sulla scheda madre. Qualora si usi una porta seriale su adattatore USB infatti questi problemi di conflitti non si pongono, dato che in tal caso vengono usate le risorse associate a questo bus di comunicazione, mantenendo per il resto inalterate le capacità di comunicazione.

Data la loro presenza fin dai primi PC compatibili, l'assegnazione delle risorse delle porte seriali sulla scheda madre è predefinita, e sia le porte di I/O che gli *interrupt* utilizzati per le 4 porte previste dall'architettura classica del PC sono riportati in tab. 5.40, insieme ai file di dispositivo con cui si può accedere ad essi.

Interrupt	Porta I/O	Dispositivo
4	`0x3F8`	`/dev/ttyS0`
3	`0x2F8`	`/dev/ttyS1`
4	`0x3E8`	`/dev/ttyS2`
3	`0x2E8`	`/dev/ttyS3`

Tabella 5.40: Risorse e file di dispositivo usati dalle porte seriali.

Benché le seriali siano praticamente scomparse dai PC ordinari, è comunque utile tenere presente i file di dispositivo di tab. 5.40, dato che queste interfacce restano presenti su moltissimi dispositivi embedded su cui gira Linux. La lettura o la scrittura dei dati di una seriale infatti avviene in maniera estremamente semplice: basta leggere o scrivere sul suo file di dispositivo.

Uno dei problemi più comuni che si possono avere utilizzando altre porte oltre quelle standard è che queste possono non essere configurate correttamente. Per ovviare a questo problema si può utilizzare il programma di configurazione delle interfacce seriali `setserial`. Questo prende come primo argomento il file di dispositivo da configurare (o controllare), se utilizzato senza opzioni viene stampato un breve riassunto delle principali caratteristiche dello stesso:

```
hain:~# setserial /dev/ttyS1
/dev/ttyS1, UART: 16550A, Port: 0x02f8, IRQ: 3
```

dove sono mostrate rispettivamente il tipo di chip della porta seriale (nel caso un `16550A`) la porta di I/O e la linea di *interrupt* utilizzate.

[52] quelli esterni, ma pure alcuni di quelli interni, sia che siano modem reali che modem finti, come i winmodem.

Opzione	Significato
-a	stampa tutte le informazioni disponibili.
-b	stampa un riassunto della configurazione.
-G	stampa le informazioni nel formato in cui vengono prese sulla riga di comando.
-z	azzera tutti i valori prima di fare le impostazioni.

Tabella 5.41: Principali opzioni per il comando `setserial`.

Quando il comando viene utilizzato con l'opzione `-g` gli argomenti vengono interpretati come una lista di dispositivi di cui stampare le proprietà; l'output delle informazioni riportate dal comando è controllato dalle ulteriori opzioni `-a`, che fa stampare tutte le informazioni possibili, e `-b` che riporta solo un riassunto della configurazione del dispositivo; l'elenco delle principali opzioni è riportato in tab. 5.41.

Se invocato senza l'opzione `-g`, gli argomenti successivi al primo permettono di indicare quale parametro del dispositivo deve essere impostato dal comando; in generale soltanto l'amministratore può modificare i parametri dell'interfaccia, ma alcuni di questi possono essere impostati anche da un utente normale.

Opzione	Significato
autoconfig	fa eseguire una auto-configurazione al kernel (la porta di I/O deve essere già impostata) con cui determinare la UART e se `auto_irq` anche la linea di *interrupt*.
auto_irq	richiede di auto-configurare anche la linea di *interrupt*.
baud_base X	imposta la frequenza delle operazioni di base, in bit per secondo, in genere vale 115200, che è il massimo raggiungibile da una seriale standard.
divisor N	imposta il divisore con cui calcolare una velocità personalizzata.
irq N	imposta il numero della linea di *interrupt*.
port 0xHHHH	imposta la porta di I/O.
spd_hi	richiede l'uso di una velocità di 57600bps (bit per secondo).
spd_vhi	richiede l'uso di una velocità di 115200bps (bit per secondo).
spd_shi	richiede l'uso di una velocità di 230400bps (bit per secondo).
spd_warp	richiede l'uso di una velocità di 460800bps (bit per secondo).
spd_normal	richiede l'uso di una velocità di 38400bps (bit per secondo).
spd_cust	richiede una velocità personalizzata pari a valore di `baud_base` diviso per quello di `divisor`.
uart type	imposta il tipo di chip, i principali valori sono `8250`, `16450`, `16550`, `16550A`, ecc. (per la lista si faccia riferimento alla pagina di manuale); con `none` si disabilita la porta.

Tabella 5.42: Principali direttive di impostazione del comando `setserial`.

I due parametri principali impostabili dal comando sono `port` e `irq`, il cui significato è ovvio e che prendono rispettivamente come ulteriore argomento il numero di porta e di *interrupt*. Si può inoltre impostare a mano (qualora non venisse riconosciuta correttamente) il tipo di interfaccia (la UART, *Universal Asynchronous Receiver/Transmitter*) con `uart`; infine si può controllare la velocità della porta seriale con la serie di argomenti `spd_*` il cui significato, insieme a quello degli altri parametri principali, è riportato in tab. 5.42; al solito per una descrizione dettagliata su può fare riferimento alla pagina di manuale di `setserial`.

Dato che spesso l'interfaccia viene usata per fornire una console di accesso (si noti come il nome del file di dispositivo richiami quello generico dell'interfaccia dei terminali `tty`) il modo più comune per usare l'interfaccia è quella di fare ricorso ad un emulatore di terminale seriale da usare interattivamente, come il programma `minicom`. In tal caso il programma, lanciato all'interno di una qualunque finestra di terminale, legge dalla seriale stampando sullo schermo e scrive sulla stessa quanto immesso dalla tastiera.

Quando viene lanciato `minicom` si collega sulla porta seriale che si è configurata come default. Questa si può cambiare lanciando il comando con l'opzione `-s` che non effettua il collegamento e fa partire il comando (che fornisce una interfaccia semigrafica) direttamente sul menu di configurazione che consente di scegliere la porta seriale e le impostazioni.

5.4.4 Gestione delle interfacce USB

L'interfaccia USB (*Universal Serial Bus*) nasce sulla piattaforma PC allo scopo di fornire una interfaccia di comunicazione semplificata e con prestazioni ridotte, ma molto semplice da realizzare e poco costosa per la realizzazione di dispositivi semplici (ed in genere portabili, è infatti prevista anche la capacità di fornire alimentazione) che non necessitano di tutte le risorse (in termini di velocità di accesso, potenza di alimentazione e banda passante nella trasmissione dei dati) fornite dalle usuali interfacce di espansione trattate in sez. 5.4.1.

Un bus USB è costruito in maniera gerarchica ed è controllato da una *unità di controllo* (o *host*), a cui è direttamente collegato un *concentratore* (o *hub*) radice a cui tutti i dispositivi o eventuali altri concentratori secondari vanno ad agganciarsi, secondo la struttura mostrata in fig. 5.7. Il protocollo prevede che le comunicazioni siano tutte controllate dall'unità di controllo, che è l'unica che può iniziare una comunicazione, ed i dispositivi possono solo rispondere all'unità di controllo; non è prevista nessuna possibilità di comunicazione dei singoli dispositivi fra di loro.

Altre architetture di bus non hanno queste limitazioni, ma USB è stato progettato con un compromesso fra prestazione ed economicità, per cui si sono evitate funzionalità avanzate come l'*arbitraggio* del bus che comportano una maggiore complessità realizzativa per dispositivi che dovevano essere estremamente semplici.

In genere su un computer è presente un solo *host*, anche se, per la diffusione di dispositivi, alcuni ne hanno più di uno, ed è comunque possibile aggiungerne altri usando schede di espansione. A ciascun *host* in genere sono agganciate 2 o 4 porte. A ciascuna porta USB si può agganciare un dispositivo o un *hub*, il quale a sua volta avrà altre porte cui agganciare altri dispositivi ed *hub*.

Il bus prevede fino ad un massimo di 127 unità per ciascuna unità di controllo; la presenza di *hub* secondari permette l'utilizzo di un maggior numero di periferiche rispetto alle porte presenti sul PC, ma si tenga conto che anche essi contano come unità sul bus, pertanto le unità disponibili per le periferiche sono in realtà 127 meno il numero di *hub* aggiuntivi.

Come mostrato in fig. 5.7 il bus prevede due direzioni per il flusso dei dati, dall'*host* alla periferica e viceversa, indicati con *downstream* e *upstream*; inoltre vengono specificati quattro *tipi* di trasferimento:

Control transfers
: utilizzati per inviare pacchetti di controllo, che devono essere di dimensioni ridotte e trasportati con affidabilità; vengono usati per configurare i dispositivi e per supportare i comandi di controllo di base.

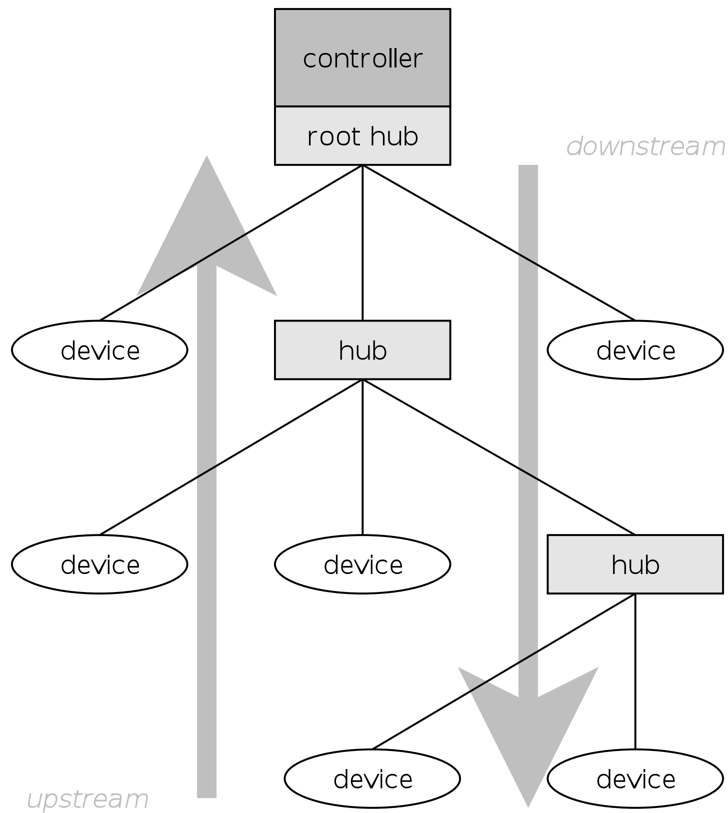


Figura 5.7: Schema della struttura di un bus USB.

Bulk transfers
usati per trasmettere dati alla massima velocità possibile in maniera affidabile; vengono usati per i trasferimenti di dati da e verso i dispositivi.

Interrupt transfers
simili ai precedenti, ma ripetuti periodicamente; la richiesta cioè viene ripetuta periodicamente e vengono utilizzati come meccanismo di notifica dato che non esiste un vero e proprio *interrupt*.

Isochronous transfers
usati per inviare e riceve dati potendo contare su una quantità di banda garantita, ma senza affidabilità, usati per trasferimenti *real-time* (ad esempio i trasferimenti per video e audio).

La presenza di due direzioni nel flusso di dati si riflette anche sui connettori, che in genere sono classificati in due tipi diversi, a seconda che siano rivolti all'unità di controllo (di *tipo A*)

o verso una periferica (di *tipo B*). Inoltre i dispositivi si possono suddividere in auto-alimentati, alimentati dal bus (che fornisce un massimo di 400 mA, quindi attenzione a non sovraccaricare, pena il rischio di non far funzionare nulla per mancanza di potenza) o entrambi. Un'altra distinzione fra i dispositivi è fra quelli *lenti* (come mouse, tastiere, ecc.) che comunicano al massimo ad 1.5MB/s e quelli *veloci* che possono usare potenzialmente fino al 90% della banda passante massima.

La prima versione del protocollo (USB 1.0) consentiva una banda passante massima di 1.5 Mbit/s ma venne presto aumentata a 12Mbit/s con la versione 1.1. Tuttavia l'uso di dispositivi lenti, gli *interrupt* e l'overhead del protocollo non consentono velocità superiori a 8.5Mbit/s anche in condizioni ideali, mentre prestazioni tipiche sono intorno ai 2Mbit/s. La seconda versione del protocollo, la USB 2.0, permette invece di portare la banda passante ad un massimo teorico 480Mbit/s, ampliando notevolmente l'utilizzabilità del bus per l'utilizzo con dispositivi di stoccaggio esterni, in particolare con dischi rimuovibili. Infine l'ultima versione del protocollo (USB 3.0) porta la banda passante a 4.8 Gbit/s (anche se quella effettiva viene stimata ad un massimo di 3.2 Gbit/s) con la modalità *superspeed* che richiede un diverso tipo di cavi, consente anche l'uso di interconnessioni ottiche pur restando retro-compatibile con USB 2.0. L'ultima versione alla scrittura di questa sezione, soltanto annunciata, è la 3.1 che arriva ad una velocità teorica di 10 Gbit/s.

Una delle caratteristiche del bus USB è che, a differenza ad esempio di quanto avveniva all'epoca per il bus SCSI, la numerazione delle periferiche presenti non deve essere effettuata a mano, essendo eseguita dal bus stesso. L'altra caratteristica interessante del bus USB è che esso supporta sempre la possibilità di disconnettere a caldo i dispositivi, anche se questo può avere conseguenze poco piacevoli se fatto senza accortezza. Per questo motivo in genere all'uso del bus è legato quello del sistema di *udev* e *hotplug* (vedi sez. 5.4.5) che è in grado di rilevare la connessione e la disconnessione di nuove periferiche caricando (e rimuovendo) automaticamente i relativi moduli.

Il bus fornisce inoltre al kernel la possibilità di rilevare tutta una serie di informazioni dai singoli dispositivi, il protocollo infatti prevede che questi forniscano i dati relativi a loro stessi. Questa informazione era inizialmente disponibile sotto la directory `/proc/bus/usb` una volta che si fosse montato il filesystem virtuale *procbususb*.

In questo caso sotto la directory `/proc/bus/usb` compariranno tante directory quanti sono i bus disponibili (numerate in ordine crescente), ciascuna delle quali conterrà un file con nome il numero associato a ciascun dispositivo presente su detto bus, contenente i relativi dati. Un sommario con una presentazione più leggibile degli stessi dati è fornito invece dal file `/proc/bus/usb/devices` che riporta l'elenco di tutti i dispositivi presenti con un contenuto del tipo:

/proc/bus/usb/devices

```
T:  Bus=01 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=480 MxCh= 6
B:  Alloc=  0/800 us ( 0%), #Int=  0, #Iso=  0
D:  Ver= 2.00 Cls=09(hub  ) Sub=00 Prot=01 MxPS= 8 #Cfgs=  1
P:  Vendor=0000 ProdID=0000 Rev= 2.06
S:  Manufacturer=Linux 2.6.6 ehci_hcd
S:  Product=VIA Technologies, Inc. USB 2.0
S:  SerialNumber=0000:00:10.3
C:* #Ifs= 1 Cfg#= 1 Atr=40 MxPwr=  0mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   2 Ivl=256ms
```

```
T:  Bus=01 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  2 Spd=480 MxCh= 0
D:  Ver= 2.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs=  1
P:  Vendor=04b4 ProdID=6830 Rev= 0.01
S:  Manufacturer=Cypress Semiconductor
S:  Product=USB2.0 Storage Device
S:  SerialNumber=600000001814
C:* #Ifs= 1 Cfg#= 1 Atr=c0 MxPwr=  0mA
I:  If#= 0 Alt= 0 #EPs= 2 Cls=08(stor.) Sub=06 Prot=50 Driver=usb-storage
E:  Ad=02(O) Atr=02(Bulk) MxPS= 512 Ivl=0ms
E:  Ad=88(I) Atr=02(Bulk) MxPS= 512 Ivl=0ms
```

Il principale utilizzo di questo file è per verificare se il riconoscimento di una periferica e la relativa numerazione è stata eseguita correttamente, infatti per ciascun dispositivo presente sul bus viene riportata nel file una diversa sezione con tutti i dati ad esso relativi. Ogni sezione è introdotta da una linea che inizia con la lettera `T` e che indica il posizionamento del dispositivo all'interno del bus, dove `Bus` indica il numero progressivo del bus, `Lev` il livello (in termini di passaggi per eventuali *hub* secondari) e `Dev` il numero progressivo assegnato.

Nelle righe successive sono poi riportate altre informazioni relative al dispositivo, sono particolarmente importanti, come riportati nella riga iniziante per `P`, gli identificatori del venditore e del prodotto che vengono utilizzati per selezionare automaticamente il modulo da utilizzare per il supporto del dispositivo, ma che in caso di fallimento possono essere utilizzati per cercare informazioni su Internet; informazioni utili relative al dispositivo sono quelle delle stringhe fornite dal produttore e mostrate nelle righe inizianti per `S`. Per il resto della sintassi si può fare riferimento alla *Linux USB Guide* disponibile su `http://www.linux-usb.org/USB-guide/book1.html`.

Oggi la stessa informazione viene presentata direttamente sotto `/sys` tramite l'uso di *sysfs* ed in genere `/proc/bus/usb` non viene più utilizzato. In tal caso però l'informazione risulta parcellizzata nelle varie sottodirectory di `/sys/bus/usb`. Per avere una visualizzazione più intelligibile delle informazioni, sono disponibili i programmi del pacchetto `usbutils`, in particolare il programma `lsusb` permette di avere una lista molto semplice dei dispositivi presenti nella forma:

```
root@hain:~# lsusb
Bus 001 Device 002: ID 04b4:6830 Cypress Semiconductor Corp.
Bus 001 Device 001: ID 0000:0000
```

Il programma può comunque stampare la lista anche in forma gerarchica usando l'opzione `-t`, mentre si possono ottenere informazioni molto più dettagliate con l'opzione `-v`. Si possono poi usare le opzioni `-s` e `-d` in maniera analoga a come si fa con `lspci` per selezionare un singolo dispositivo sulla base rispettivamente del suo bus e numero identificativo (con la sintassi `bus:num`) o del produttore (usando i rispettivi identificatori cui abbiamo accennato in precedenza). Al solito si faccia riferimento alla pagina di manuale per una descrizione completa.

5.4.5 La gestione dei dispositivi con *udev* e *hotplug*

Come accennato in sez. 5.2.5 nello sviluppo del kernel 2.6 è stata completamente ristrutturata la gestione dei moduli come conseguenza della creazione di una rappresentazione generale dei dispositivi (il cosiddetto *device model*) da parte del kernel con l'uso del filesystem *sysfs* e della directory `/sys`. In concomitanza a questo è stato poi sviluppato, in considerazione del sempre

maggiore utilizzo di Linux in ambienti di tipo desktop, un meccanismo di gestione il più trasparente e semplificato possibile per automatizzare rilevamento e supporto dei vari dispositivi, sia fissi che rimuovibili.

In particolare uno dei problemi affrontati è stato quello dei dispositivi rimuovibili a caldo, sempre più diffusi anche nelle macchine comuni anche grazie a bus come l'USB o il Firewire. Per questo sono state realizzate diverse infrastrutture, in particolare fin dal kernel 2.4 era stato introdotto un meccanismo, denominato *hotplug*, per cui in reazione alla avvenuta presenza di dispositivi rimuovibili il kernel stesso provvedeva all'esecuzione di un apposito programma in user space, `/sbin/hotplug`,[53] che si incaricava di caricare gli opportuni moduli ed effettuare tutte le operazioni reputate necessarie alla gestione del dispositivo.

Il meccanismo classico di *hotplug* prevedeva il passaggio delle informazioni necessarie al programma tramite delle opportune variabili di ambiente; queste poi venivano analizzate dal programma per identificare il dispositivo che aveva generato l'evento, ed eseguire le azioni necessarie alla sua gestione, fra cui ovviamente il caricamento dei moduli per il relativo supporto. Tutte queste operazioni venivano realizzate tramite una serie di script di shell mantenuti in `/etc/hotplug.d`, e suddivisi per i vari tipi di bus.

Con lo sviluppo dei kernel della serie 2.6.x questo meccanismo ha cominciato a mostrare i suoi limiti. Da una parte infatti si erano ampliate le richieste di notifiche di eventi non più semplicemente legati alla disponibilità di un nuovo dispositivo, ad esempio viene generato un evento anche in risposta all'inserimento di un cavo in una scheda di rete, così da poterla configurare automaticamente. Dall'altra lo sviluppo della rappresentazione generica dei dispositivi richiedeva la generazione di eventi tutte le volte che si doveva registrare una qualche nuova informazione. L'impatto dell'analisi di tutti questi eventi iniziava ad essere pesante.

Allo stesso tempo i kernel della serie 2.6.x avevano visto una completa ristrutturazione della infrastruttura per la creazione dinamica dei file di dispositivo (dinamica nel senso che prevedesse la presenza in `/dev` soltanto dei file relativi ai dispositivi effettivamente presenti, e non di tutti quelli possibili) con una realizzazione interamente in user space che rimpiazzasse il precedente *devfs*,[54] La nuova infrastruttura, denominata *udev*, prevedeva l'uso delle informazioni mantenute nell'apposito filesystem *sysfs* e degli eventi di *hotplug* per la creazione automatica dei file di dispositivo sotto `/dev`, ed un insieme di "*regole*" in grado di governare non solo i nomi, ma anche i permessi e le altre caratteristiche da assegnare ai singoli dispositivi.

Per un certo periodo i due sistemi, *udev* ed *hotplug*, sono stati sviluppati indipendentemente, ma nel corso della ristrutturazione di *hotplug* per far fronte al crescente numero di eventi gli sviluppatori si sono resi conto che dato che *udev* doveva comunque analizzare tutti gli eventi per creare i file di dispositivo, non aveva senso usare un programma a parte per caricare i moduli necessari, e che questo compito poteva essere eseguito direttamente dal sistema di *udev*.

Questo risultava possibile anche grazie alla ristrutturazione della gestione dei moduli stessi. Come abbiamo visto in sez. 5.2.5 ogni modulo porta con sé, fra i parametri ad esso associati, una lista degli *alias* che indicano a quale tipo di dispositivo esso può venire associato; quando il kernel

[53] o qualunque altro programma fosse stato indicato nel file `/proc/sys/kernel/hotplug`.

[54] questo era un filesystem virtuale che creava automaticamente il contenuto di `/dev`, ma oltre a non essere più mantenuto e presentare notevoli problemi nella realizzazione, costringeva ad inserire le politiche relative alla gestione dei dispositivi (i permessi ad esempio) all'interno del kernel; per questo è stato sostituito ed il relativo supporto è stato completamente rimosso a partire dal kernel 2.6.13.

riconosce la presenza di un nuovo dispositivo è suo compito generare un *alias* per lo stesso,[55] che sarà quello usato da *udev* per caricare il relativo modulo. Questo è quanto avviene oggi, a lungo, fino al 2.6.15, il meccanismo prevedeva che il kernel creasse delle variabili di ambiente che poi dovevano essere esaminate da *hotplug* per identificare il modulo.

Per questo motivo tutte le distribuzioni recenti non prevedono più l'uso degli script di *hotplug* e la directory `/etc/hotplug.d` è sostanzialmente vuota se non assente. Il caricamento dei moduli viene infatti gestito direttamente dal sistema di *udev* per qualunque dispositivo venga rilevato dal kernel, mentre le eventuali operazioni da compiere per gli eventi relativi a dispositivi rimuovibili vengono gestite in genere passando tutti gli eventi ad un apposito demone.

Questo cambia anche le modalità con cui viene eseguito il caricamento dei moduli, in precedenza infatti questo era associato al primo accesso ad un dispositivo, adesso invece è associato alla sua rilevazione, è per questo motivo che se non si vuole che certi moduli non siano caricati a meno che non li si utilizzi, questi deve essere impedito esplicitamente: si riveda quanto detto riguardo la direttiva `blacklist` di `modprobe.conf` in sez. 5.2.5.

Ovviamente è possibile evitare l'uso di *udev*, creando un contenuto statico per `/dev` e mantenendo nel kernel il supporto per i dispositivi essenziali (è quello che viene fatto per i *ramdisk* di avvio), ma l'infrastruttura si è dimostrata estremamente flessibile, ed è stata adottata da tutte le distribuzioni. La flessibilità di *udev* deriva dal fatto che la creazione dei file di dispositivo sotto `/dev` può essere controllata attraverso una serie di "*regole*", che consentono non solo di specificarne le proprietà come nome, permessi, proprietario, ecc. ma anche di associare alla creazione stessa azioni ulteriori, come la creazione di più nomi o di link simbolici, e l'esecuzione di ulteriori programmi.

Come accennato *udev* si appoggia al nuovo filesystem *sysfs*, introdotto con i kernel della serie 2.6.x. Si tratta di un filesystem virtuale analogo a `/proc`, il cui scopo principale è quello di rendere visibile le informazioni presenti nel kernel riguardo dispositivi e relativi moduli (il cosiddetto *device model* del kernel) ai programmi in user-space. Come richiesto dal *Filesystem Hierarchy Standard* (vedi sez. 1.2.3) deve essere montato sotto `/sys`. In questo caso la richiesta è più stringente di una semplice esigenza di standardizzazione: tutti i programmi e le utilità di basso livello fanno l'assunzione che *sysfs* sia montato sotto `/sys`, e non funzionano se lo si sposta altrove.

In sostanza l'idea sottostante l'uso di *sysfs* era quella di esporre in questo filesystem caratteristiche e proprietà dei dispositivi e sottosistemi gestiti dal kernel e di fornire qui la possibilità di effettuare eventuali impostazioni, cercando di riportare `/proc` al suo scopo originale di fornire informazioni relative ai processi.

Il vantaggio di *sysfs* è che le informazioni sui dispositivi ed i moduli vengono generate direttamente all'origine dal kernel stesso, e non sono necessarie ulteriori intermediazioni; ogni volta che il kernel riconosce un nuovo oggetto all'interno del suo *device model* la relativa informazione viene creata automaticamente dentro questo filesystem.

In particolare sotto `/sys/devices` viene creata automaticamente la gerarchia generale di tutti i dispositivi rilevati dal kernel, espressa come un albero di directory che rappresentano le dipendenze reciproche fra gli stessi. Così ad esempio un dispositivo PCI (come un controller SATA) sarà rappresentato da una directory inserita sotto la directory relativa al bus PCI su cui

[55] il kernel lo ricava dalle informazioni ottenute dal dispositivo stesso, grazie agli identificativi previsti nei vari bus (si ricordi quanto appena visto per PCI e USB).

esso si trova, ed un disco attaccato a un controller sarà inserito in una directory sotto quella del suo controller.

Inoltre sotto `/sys/devices` vengono rappresentati anche quei dispositivi che non corrispondono ad una periferica (ad esempio la CPU) ma che sono parte integrante del sistema (sotto `/sys/devices/system`) e quelli specifici di alcune piattaforme hardware che non rientrano in nessuna gerarchia, come le porte di I/O usate per la seriale che si trovano solo sui PC (sotto `/sys/devices/platform`).

Oltre a questo il sistema prevede la generazione di una serie di informazioni aggiuntive. Ad esempio la directory `/sys/bus` viene automaticamente popolata con link simbolici agli stessi dispositivi classificati secondo i bus che sono stati registrati come tali nel kernel, mentre la directory `/sys/class` conterrà invece la classificazione dei dispositivi secondo le relative classi (di nuovo queste vengono definite direttamente dal kernel e gestite al suo interno), e `/sys/block` l'elenco dei dispositivi a blocchi. Inoltre la directory `/sys/modules` conterrà una directory per ciascun modulo caricato dal kernel con le relative informazioni.

Un'altra delle caratteristiche distintive di *sysfs* è che è possibile associare in maniera molto semplice a ciascun dispositivo o modulo degli "*attributi*" che verranno visualizzati nel filesystem come file, su cui è possibile leggere (e anche scrivere). In questo modo diventa semplice rendere disponibili informazioni o caratteristiche (eventualmente modificabili per effettuare impostazioni) relative agli stessi.

Con lo sviluppo del kernel il contenuto si *sysfs* si è ampliato molto, sono stati aggiunti contenuti e molte proprietà dei dispositivi ed altre funzionalità del kernel possono essere impostate direttamente attraverso i file presenti in `/sys`, con l'indicazione da parte dagli sviluppatori del kernel di spostare al suo interno tutte quelle funzionalità che in precedenza sarebbero state gestite tramite `/proc`, che dovrebbe invece tornare ad essere un filesystem dedicato alla gestione dei processi. Questo però non significa che *sysfs* abbia completamente sostituito `/proc` ed in particolare `/proc/sys` continua a mantenere lo stesso significato illustrato in sez. 3.1.3.

La creazione dinamica del contenuto di `/dev` viene realizzata tramite il demone `udevd`, che viene lanciato all'avvio e riceve dal kernel gli eventi di notifica per ciascun dispositivo che viene aggiunto o rimosso dal sistema o che cambia stato. È compito di `udevd` identificare il tipo di dispositivo utilizzando sia le informazioni ricevute con l'evento che quelle presenti sotto `/sys`, e creare di conseguenza i file di dispositivo sotto `/dev` che viene creata come directory posta su un filesystem temporaneo (la si può anche osservare nell'output di `df` mostrato in sez. 5.1.3).

Il comportamento di base del sistema di *udev* è controllato dal relativo file di configurazione `/etc/udev/udev.conf`; questo usa la sintassi di un file di assegnazioni di variabili di shell. In genere lo si usa per impostare alcune delle variabili chiave che consentono di soprassedere i valori di default usati dal sistema. Le sole chiavi che può valer la pena di modificare sono `udev_log`, che controlla la priorità dei messaggi inviati al sistema di *syslog* (con i valori di tab. 3.5) e `tmpfs_size` che controlla le dimensioni del filesystem temporaneo su cui è creata `/dev`, un contenuto tipico è il seguente (estratto da una Debian Lenny):

/etc/udev/udev.conf

```
# The initial syslog(3) priority: "err", "info", "debug" or its
# numerical equivalent. For runtime debugging, the daemons internal
# state can be changed with: "udevadm control --log_priority=<value>".
udev_log="err"
# maximum size of the /dev tmpfs
tmpfs_size="10M"
```

Oltre a detto file di configurazione generale le funzionalità di *udev* vengono controllate grazie all'uso di appropriate "*regole*" con cui si esprimono le operazioni da effettuare in conseguenza ai vari eventi che `udevd` riceve dal kernel. Alla ricezione di ciascun evento `udevd` processa le regole ed esegue le operazioni in esse indicate, inoltre tutta l'informazione viene comunque memorizzata in un database e può essere acceduta da altri programmi.

Le regole devono essere inserite in appositi file con l'estensione `.rules` (altre estensioni sono ignorate), mantenuti normalmente sotto `/etc/udev/rules.d`, che è la directory riservata agli utenti per la configurazione delle regole; è possibile, anche se non c'è nessun motivo per farlo, cambiare questa directory con la direttiva `udev_rules` in `/etc/udev/udev.conf`. Se presenti però vengono utilizzate anche le regole predefinite di sistema poste in `/lib/udev/rules.d`, ed eventuali regole temporanee poste in `/dev/.udev/rules.d`.

Quando `udevd` parte tutti i file delle regole vengono letti e processati in ordine alfabetico, indipendentemente dalle directory in cui si trovano, ma in caso di uguaglianza dei nomi hanno la precedenza i file presenti in `/etc/udev/rules.d`. Questo significa che se ci sono regole da eseguire per prime queste devono essere inserite in file il cui nome abbia la precedenza, questo è il motivo per cui i file di regole di `/etc/udev/rules.d` hanno normalmente nomi che iniziano con dei codici numerici, così da identificare in maniera più immediata la sequenza in cui verranno utilizzati.

La sintassi con cui si esprimono le regole è abbastanza complessa; ogni riga del file, a parte quelle vuote ed i commenti introdotti dal solito carattere "`#`", deve contenere almeno una coppia chiave/valore; è poi possibile specificarne anche più di una con una lista separate da virgole. Esistono due tipi di chiavi, le chiavi di confronto (dette *matching key*) usate per selezionare gli eventi a cui applicare la regola e quelle di assegnazione (dette *assign key*) con cui si controllano le operazioni da eseguire.

Operatore	**Significato**
`==`	confronta l'eguaglianza di una *matching key*.
`!=`	confronto la diseguaglianza di una *matching key*.
`=`	assegna un valore a una *assign key*.
`+=`	aggiunge un valore a una *assign key*.
`:=`	assegna definitivamente un valore a una *assign key*.

Tabella 5.43: Operatori delle regole di *udev*.

Ciascun tipo di chiave prevede diversi operatori con cui essa viene associata al proprio valore nella forma di una assegnazione in cui il valore viene espresso come stringa fra apici doppi. Gli operatori utilizzabili sono riportati in tab. 5.43, nella parte superiore ci sono gli operatori di confronto, il cui uso identifica una *matching key*, nella parte inferiore quelli di assegnazione, il cui uso identifica una *assign key*. Si tenga presente inoltre che nelle espressioni dei valori usati con le *matching key* è generalmente supportato l'uso degli operatori del *filename globbing* ("*", "`?`" e "`[]`"), con lo stesso significato che hanno nella shell.

Si tenga presente inoltre che alcune chiavi, a seconda dell'operatore che vi si applica, possono essere usate sia per confronto o per assegnazione; un elenco delle chiavi di confronto più significative è riportato in tab. 5.44 mentre in tab. 5.45 si sono riportate quelle di assegnazione. Per l'elenco completo ed i dettagli si consulti la pagina di manuale di *udev* con `man udev`.

Chiave	Significato
ACTION	nome dell'azione riportato dell'evento.
ATTR{name}	valore dell'attributo name associato su *sysfs* al dispositivo dell'evento.
KERNEL	nome del dispositivo secondo il kernel, come riportato nell'evento.
NAME	nome dell'interfaccia di rete o del file di dispositivo sotto /dev, usabile se assegnato in una regola precedente.
SUBSYSTEM	nome del sottosistema riportato dall'evento.

Tabella 5.44: Chiavi di confronto per le regole di *udev*.

Chiave	Significato
GROUP	gruppo proprietario del file di dispositivo.
MODE	permessi del file dispositivo.
NAME	nome dell'interfaccia di rete o del file di dispositivo sotto /dev.
OWNER	proprietario del file di dispositivo.
RUN	programma da eseguire.

Tabella 5.45: Chiavi di assegnazione per le regole di *udev*.

Se in corrispondenza ad un evento tutte le *matching key* presenti in una riga corrispondono, verranno assegnati i valori alle *assign key* presenti nella stessa; questo, a seconda di quali chiavi si sono usate, può comportare l'esecuzione di varie operazioni, come la creazione di un file di dispositivo, o di un link simbolico, fino anche alla esecuzione di un programma ad hoc.

Come esempio di un file di regole possiamo prendere in esame il seguente contenuto del file `z60_hdparm.rules`, installato con il pacchetto Debian del programma `hdparm` (vedi sez. 6.3.1):

z60_hdparm.rules
```
ACTION=="add", SUBSYSTEM=="block", KERNEL=="[sh]d[a-z]", RUN+="/etc/init.d/hdparm hotplug"
```

In questo caso nel caso che il kernel riporti l'aggiunta di un dispositivo a blocchi il cui nome corrisponda a quelle di un disco, si richiede l'aggiunta alla lista dei programmi da eseguire del lancio dello script di avvio di `hdparm` così che questo possa tener conto del nuovo disco.

Per controllare il comportamento di *udev* erano stati forniti inizialmente alcuni programmi singoli (`udevinfo` e `udevmonitor`) le cui funzionalità sono state in seguito raccolte in un unico programma, `udevadm` che consente di operare sul sistema. Il comando richiede che si specifichi come argomento un sotto-comando indicante il tipo di operazioni che si vogliono effettuare, che vengono specificate ulteriormente tramite le opportune opzioni.

In particolare con `udevadm info`, che sostituisce `udevinfo`, si possono ottenere le informazioni presenti nel database di *udev* relative ad un qualunque dispositivo riconosciuto dal kernel. Il comando richiede che si indichi il dispositivo di cui si richiedono le informazioni e questo può essere fatto in due modi; il primo è tramite l'opzione `-p` (o `--path`) che consente di specificarlo tramite il *pathname* con cui esso compare sotto `/sys/device`, ma in genere, dato che detto pathname può risultare molto complesso per via della gerarchia dei dispositivi, si utilizza molto più spesso l'opzione `-n` (o `-name`), che prende invece il nome del dispositivo come compare sotto `/dev`. La prima opzione però, al costo di una identificazione più complicata, consente di ottenere

le informazioni anche quando per un qualche malfunzionamento delle regole di *udev* non viene creato un file di dispositivo sotto `/dev`.

Le richieste di informazione vengono controllate dall'opzione `-q` (o `--query`) che prende un argomento e consente di specificare le informazioni che si richiedono, il cui possibile valore è illustrato in tab. 5.46, in genere si usa `path` per ottenere il pathname del dispositivo sotto `/sys/device`.

Valore	**Significato**
`name`	file di dispositivo sotto `/dev`.
`symlink`	link simbolici relativi al dispositivo.
`path`	*pathname* sotto `/sys`.
`property`	proprietà del dispositivo.
`all`	tutte le precedenti.

Tabella 5.46: Possibili valore per l'opzione `-q` di `udevadm info`.

Inoltre, se si usa l'opzione `-a` (o `--attribute-walk`) invece di `-q` è possibile ottenere i valori di tutti gli attributi associati al dispositivo (e ai bus o dispositivi di cui fa parte nella gerarchia di *sysfs*), vale a dire quelli controllabili con le *matching key* di tipo `ATTR{name}`.

Una altro sotto-comando molto utile, specie quando si deve effettuare una diagnosi di funzionamento di *udev*, è `udevadm monitor`, (che sostituisce `udevmonitor`) che pone sotto osservazione il sistema e stampa sulla console i dati relativi sia agli eventi inviati dal kernel che a quelli generati dal sistema stesso. In questo caso esistono varie opzioni che consentono di controllare e filtrare l'output del comando, ad esempio con `--kernel` si può richiedere la stampa degli eventi del kernel, con `--udev` di quelli interni ad *udev*, o usare `--subsystem-match` per chiedere (tramite la stringa che lo identifica passata come parametro) la stampa dei soli eventi relativi ad un sottosistema (quelli di `/sys/subsystem`).

Sotto-comando	**Significato**
`control`	esegue operazioni di controllo su `udevd`.
`help`	stampa l'elenco dei sotto-comandi.
`info`	ricerca informazioni nei dati di *sysfs* e *udev*.
`monitor`	si pone in ascolto degli eventi e stampa informazioni a video.
`test`	simula un evento.

Tabella 5.47: Sotto-comandi del comando `udevadm`.

In tab. 5.47 si sono riportati i principali sotto-comandi di `udevadm`. Un elenco completo si può ottenere con `udevadm help`, e per le informazioni dettagliate riguardo a tutte le opzioni disponibili per ciascuno di essi si rimanda alla pagina di manuale, accessibile al solito con `man udevadm`.

Capitolo 6

Amministrazione avanzata del sistema

6.1 L'utilizzo del RAID

Tratteremo in questa prima sezione l'utilizzo dei sistemi RAID, una tecnologia che permette di utilizzare i dischi a gruppi (detti *array*) per ottenere migliori prestazioni o una maggiore affidabilità; partiremo con una introduzione teorica a questa tecnologia per poi vedere le varie modalità con cui essa viene realizzata e può essere utilizzata con Linux.

6.1.1 Introduzione

La tecnologia RAID, acronimo che sta per *Redundant Arrays of Inexpensive Disks*, nasce con un articolo pubblicato nel 1987 a Berkeley, dove si descrivevano varie tipologie di *disk array*. L'idea di base è quella di combinare più dischi indipendenti di piccole dimensioni per ottenere o prestazioni, o affidabilità o quantità di spazio disco superiori a quelle ottenibili con un qualunque disco SLED (*Single Large Expensive Drive*).

L'articolo presentava cinque diversi tipi di architetture RAID (da RAID-1 a RAID-5) ciascuna delle quali prevedeva diversi gradi di ridondanza per fornire tolleranza alla rottura e diversi gradi di prestazioni. A queste si è aggiunta poi la notazione RAID-0 per indicare una configurazione senza ridondanza.

Oggigiorno alcune delle configurazioni previste nell'articolo non sono più supportate (se non in sistemi specializzati), e non le prenderemo neanche in considerazione, ne sono poi emerse altre come la combinazione del RAID-0 con il RAID-1, denominata RAID-10, o quella definita *linear*. Con il kernel 2.6.x inoltre è stato introdotto anche il RAID-6 analogo al RAID-5 ma in grado di assicurare una maggiore affidabilità. In generale comunque ci si suole riferire a queste configurazioni con il nome di *livelli*, quelli che prenderemo in esame sono pertanto:

linear In questo livello due o più dischi vengono combinati in un singolo dispositivo. I dischi sono semplicemente accodati uno sull'altro, così che scrivendo in maniera

lineare prima verrà riempito il primo e poi i successivi. La dimensione dei singoli dischi in questo caso è del tutto irrilevante.

Questo livello non assicura nessun tipo di ridondanza, e se uno dei dischi si rompe si perderanno probabilmente tutti i dati. Data la modalità di scrittura comunque si perderanno solo i dati presenti nel disco rotto, e qualche forma di recupero potrebbe comunque essere possibile.

Questa configurazione non migliora le prestazioni di lettura o scrittura sulla singola operazione, ma quando più utenti accedono all'array si può avere una distribuzione del carico sui vari dischi, qualora per un caso fortuito vengano eseguiti degli accessi a dati posti su dischi diversi.

RAID-0 Questo livello è anche chiamato *stripe mode*; in questo caso i dischi dovrebbero avere la stessa dimensione, anche se non è strettamente necessario. In questo caso le operazioni sull'array vengono distribuite sui singoli dischi. Se si scrive un blocco di dati questo verrà suddiviso in *strisce* (le *stripes*, appunto) di dimensione predefinita che vengono scritte in sequenza su ciascuno dei vari dischi che compongono l'array: una striscia sul primo, poi sul secondo, ecc. fino all'ultimo per poi ricominciare con il primo. La dimensione delle strisce di norma viene definita in fase di creazione dell'array, ed è uno dei parametri fondamentali per le prestazioni dello stesso. La capacità totale resta la somma di quella dei singoli dischi.

In questo modo le prestazioni di lettura e scrittura possono essere notevolmente aumentate, in quanto le operazioni vengono eseguite in parallelo su più dischi, e se i dischi sono veloci ed in numero sufficiente si può facilmente saturare la capacità del bus, o se quest'ultimo è veloce si può ottenere una banda passante totale pari alla somma di quella dei singoli dischi.

Se è presente un disco di dimensioni maggiori degli altri si potrà continuare ad accedere operando solo sullo spazio presente sulla parte finale dello stesso, ma in questo caso si opererà su un solo disco, e non si avrà quindi nessun miglioramento delle prestazioni per quanto riguarda i dati che vanno a finire su di esso.

Come nel caso precedente questo livello non fornisce nessuna ridondanza. La rottura di un disco comporta la perdita totale di tutti i dati, compresi pure quelli sugli altri dischi, dato che in questo caso non esistono, a differenza di *linear*, blocchi di dati continui più lunghi della dimensione di una striscia. In questa modalità l'MTBF (sigla che sta per *Mean Time Between Failure*) diminuisce proporzionalmente al numero di dischi dell'array in quanto la probabilità di fallimento dell'array è equivalente a quella di un singolo disco moltiplicata per il numero dei dischi che compongono l'array.

RAID-1 Questo è il primo livello che fornisce della ridondanza. Richiede almeno due dischi per l'array e l'eventuale presenza di dischi di riserva. Inoltre i dischi devono essere tutti uguali, se sono diversi la dimensione dell'array sarà quella del più piccolo dei dischi. I dati vengono scritti in parallelo su tutti i dischi che compongono l'array, pertanto basti che sopravviva anche un solo disco dell'array perché i dati siano intatti, in quanto ciascun disco ne ha una copia completa. Gli eventuali dischi di

riserva vengono utilizzati in caso di fallimento di un disco dell'array, per dare inizio ad una ricostruzione immediata dell'array stesso.

Con questo livello di RAID si aumenta l'affidabilità in quanto la probabilità di fallimento dell'array è equivalente a quella di un singolo disco divisa per il numero dei dischi dell'array. In realtà poi è ancora minore, in quanto la probabilità che i dischi falliscano tutti insieme è ancora minore, per cui basta sostituire un disco rotto per recuperare l'intera funzionalità dell'array.

Come controparte per l'affidabilità le prestazioni di scrittura di un RAID-1 sono in genere peggiori di quelle del singolo disco, in quanto i dati devono essere scritti in contemporanea su più dischi, e se questi sono molti (e si usa una implementazione software) facendo passare tante copie si può superare il limite di banda del bus su cui sono posti i dischi abbastanza facilmente. In questo caso i controller hardware hanno il vantaggio di gestire internamente la replicazione dei dati, richiedendo l'invio su bus di una sola copia.

Le prestazioni in lettura sono invece migliori, non tanto sulla singola lettura, ma quando ci sono molte letture contemporanee e spostamenti sui dati. In tal caso infatti i dati possono essere letti in parallelo da dischi diversi, e si può approfittare del diverso posizionamento delle testine sui dischi per leggere i dati da quello che deve compiere un minore spostamento delle testine.[1]

RAID-4 Benché disponibile è senz'altro il meno usato. Necessita di tre o più dischi. Mantiene delle informazioni di recupero su un disco di *parità*, ed utilizza gli altri in RAID-0. I dischi devono essere di dimensioni identiche, altrimenti l'array avrà la dimensione del più piccolo di essi. Se uno dei dischi in RAID-0 si rompe è possibile utilizzare le informazioni di recupero del disco di *parità* per ricostruire i dati in maniera completa. Se si rompono due dischi si perderanno invece tutti i dati.

Questo livello cerca di bilanciare i vantaggi di affidabilità del RAID-1 con le prestazioni del RAID-0, ma non viene usato molto spesso però perché il disco di parità viene a costituire un collo di bottiglia in quanto comunque tutte le informazioni devono esservi replicate, l'unico caso di impiego reale è quello di un disco veloce affiancato ad un gruppo di dischi più lenti.

RAID-5 Come per il RAID-4 anche in questo caso si cercano di ottenere sia vantaggi di affidabilità che quelli di prestazioni, ed è il più usato quando si hanno più di due dischi da mettere in RAID. Anche per il RAID-5 sono necessari almeno tre dischi, ma in questo caso, per evitare il problema del collo di bottiglia, le informazioni di *parità*, queste vengono distribuite su tutti i dischi che fanno parte dell'array. In questo modo anche se uno dei dischi si rompe le informazioni di parità presenti sugli altri permettono di mantenere intatti i dati. Inoltre se si sono inseriti dei dischi di riserva la ricostruzione del disco rotto o indisponibile viene fatta partire immediatamente. Se però si rompono due dischi i dati sono perduti.

[1] è caratteristica comune degli hard disk avere tempi di risposta nettamente diversi per la lettura sequenziale di dati, rispetto al posizionamento (il cosiddetto *seek time*).

Il RAID-5 presenta inoltre, dato che dati sono comunque distribuiti su più dischi, gli stessi vantaggi nella lettura presentati dal RAID-0. In scrittura invece le prestazioni sono meno predicibili, e le operazioni possono essere anche molto costose, quando richiedono pure le letture necessarie al calcolo delle informazioni di parità, o dell'ordine di quelle del RAID-1. In generale comunque le prestazioni aumentano sia in lettura che in scrittura. Inoltre nel caso di RAID software è richiesto un certo consumo di risorse (sia di memoria che di CPU) per il calcolo delle informazioni di parità.

RAID-10 Si suole definire così una configurazione in cui si effettua un RAID-0 di più array configurati in RAID-1. In questo modo si ottiene sia la ridondanza del RAID-1 che l'aumento di prestazioni del RAID-0. Rispetto al RAID-5 ha il vantaggio che possono rompersi anche più dischi, ma fintanto che almeno ne resta almeno uno attivo nei due RAID-1 le informazioni saranno integre. Il tutto al prezzo di un numero molto maggiore di dischi per ottenere la stessa capacità.

RAID-6 È concettualmente identico al RAID-5, di cui presenta le stesse caratteristiche funzionali, ma prevede che le informazioni di parità siano distribuite su due dischi aggiuntivi, così che sia possibile sopportare la rottura contemporanea di due dischi anziché uno solo.[2] Rispetto al RAID-5 ha il vantaggio di una maggiore ridondanza dato che possono rompersi anche due dischi in contemporanea, mantenendo l'integrità dei dati fintanto che gli altri restano attivi. Il tutto al prezzo di dover usare un disco in più rispetto al RAID-5 per ottenere la stessa capacità.

Un altro concetto relativo alla gestione del RAID, che si applica in genere ai livelli che forniscono ridondanza, come RAID-1, RAID-5 e RAID-6, è quello dei cosiddetti *dischi di riserva*, o *spare disk*. In questo caso si tratta di dischi che possono essere associati ad un array RAID senza però venirne a far parte come membri attivi. Il loro scopo, come dice il nome, è quello di essere usati come riserva quando uno dei dischi attivi si dimostra difettoso. In questo caso è previsto che la implementazione del RAID si accorga del problema ed inizi immediatamente ed in maniera automatica la ricostruzione dell'array utilizzando il disco di riserva al posto di quello difettoso, così da aumentare ulteriormente la salvaguardia dei dati.

Al giorno d'oggi lo scopo più comune dell'uso di un RAID è comunque quello di proteggersi dal fallimento o dalla rottura di un disco mantenendo automaticamente la ridondanza delle informazioni. Per questo il RAID-0 non deve mai essere usato in un sistema di produzione, perché il rischio di fallimento hardware si moltiplica per il numero di dischi, e con questo quello della perdita completa dei dati, cosa che normalmente non vale il limitato guadagno di prestazioni che si potrebbe ottenere a meno che non si tratti di dati mantenuti altrove e per i quali si deve fornire semplicemente un accesso il più rapido possibile.

Si tenga presente infine che l'utilizzo di un livello di RAID che assicuri la ridondanza non è mai un sostituto di una buona politica di backup. Un array RAID infatti può proteggere dal fallimento dell'hardware di uno o più dischi, ma non può nulla contro la corruzione di un

[2] evento che si è dimostrato tutt'altro che improbabile; i dischi di uno stesso produttore infatti, specie se provenienti dalla stessa catena di produzione, hanno dimostrato di avere una durata molto simile, cosa che comporta assai frequentemente delle rotture se non contemporanee comunque molto ravvicinate fra loro, con il rischio di non fare in tempo a sostituire un disco rotto prima che si rompa anche il secondo.

filesystem, contro un utente che cancella (intenzionalmente o meno) i dati, o contro un qualunque altro incidente che distrugga l'intero array.

6.1.2 Il RAID su Linux

In generale per poter creare un array di dischi in RAID ci sono due opzioni fondamentali: RAID hardware o RAID software. Nel primo caso occorre poter disporre di un supporto hardware, cioè di un controller per i dischi che esegue le operazioni necessarie alla realizzazione del RAID al suo interno, e presenti poi al sistema operativo l'insieme dei dischi come un dispositivo unico.

In teoria un RAID hardware è l'opzione migliore sul piano delle prestazioni in quanto non necessita di utilizzare delle risorse (CPU e memoria) della macchina per gestire il RAID e consente di eseguire il trasferimento di dati una volta sola (dal sistema al dispositivo hardware). In genere però richiede parecchie risorse (specie per i controller RAID SCSI di fascia alta) dal punto di vista del portafoglio.

Inoltre la superiorità di prestazioni in certi casi è solo teorica, molti controller hardware (specie quelli IDE o serial ATA disponibili a costi limitati) si sono rivelati essere molto più lenti dell'implementazione del RAID software di Linux, sia per la scarsa potenza dei processori utilizzati che per la scarsa qualità del software della scheda.[3] Questo, unito al fatto che in molti casi nei server la CPU è una risorsa meno rilevante rispetto alla stabilità e alla sicurezza dell'I/O su disco, ha fatto sì che l'opzione del RAID software sia estremamente competitiva nella maggior parte dei casi.

A questo si aggiunge poi il problema che alcuni controller hardware usano implementazioni proprietarie nella gestione dei dati del RAID e di come questi sono salvati su disco. Questo finisce poi col rendere impossibile riutilizzare i dati contenuti sugli stessi dischi su controller di marca diversa, e talvolta anche su versioni diverse di schede della stessa marca. Tutto ciò non è ovviamente mai vero nel caso del RAID software fatto dal kernel, che è in grado di riutilizzare i dischi qualunque sia la scheda su cui questi sono montati.

In ogni caso Linux supporta vari controller RAID hardware, sia SCSI che IDE che Serial ATA. Nel caso di controller SCSI in molti casi è cura del controller stesso presentare tutto l'array RAID come un singolo disco, che pertanto sarà visto semplicemente con uno dei nomi associati ai dischi SCSI (come `/dev/sda`). In tal caso le operazioni di gestione del RAID sono demandate al firmware della relativa scheda e dipendono ovviamente da quest'ultima, per cui in genere è necessario effettuare la configurazione del RAID all'avvio della macchina operando all'interno del BIOS con l'interfaccia messa a disposizione dalla scheda stessa.

Per alcuni controller RAID dedicati esistono dei driver specifici disponibili nella sezione `Block Devices` della configurazione del kernel. In questo caso i dispositivi RAID sono in genere accessibili attraverso opportune sottodirectory o file di dispositivo sotto `/dev` che dipendono dal tipo di controller. Ad esempio per il DAC Mylex viene usata una directory `rd` mentre per gli array Compaq vengono usate, a seconda del tipo le directory `ida` e `cciss`. In questi casi i singoli array vengono identificati per "*canale*" (spesso i controller supportano più di una singola catena SCSI) e per "*disco*", un dispositivo tipico di un array è sempre nella forma `/dev/rd/c0d0` che indica il primo array sul primo canale. Una volta partizionato l'array (che a questo punto viene visto come un altro disco) le partizioni saranno accessibili con nomi del tipo `/dev/ida/c0d0p1`.

[3] cosa che talvolta li rende anche inaffidabili; questo non è vero in genere per i controller SCSI di fascia alta, che hanno però anche un livello di costi totalmente diverso.

Nel caso di RAID su IDE non ci sono canali, ed i dispositivi sono accessibili, una volta attivato il relativo supporto, che è nella sezione `ATA/IDE/MFM/RLL support`, attraverso i file di dispositivo generici disposti nella directory `/dev/ataraid`, i cui nomi sono analoghi ai precedenti, soltanto che non presentano un numero di canale; si avrà cioè ad esempio qualcosa del tipo di `/dev/ataraid/d0p1`. Nel caso di Serial ATA di nuovo alcuni controller consentono di utilizzare una forma di RAID hardware presentando al kernel l'array RAID come un singolo dispositivo; di nuovo verranno visti come un disco SCSI. Si tenga presente che in molti casi le prestazioni e l'affidabilità di queste schede a basso costo si sono dimostrate molto limitate, tanto da sconsigliarne comunque l'uso a favore del supporto software presente nel kernel.

Per quando non si dispone di un supporto hardware Linux fornisce un supporto software, attivabile nella sezione `Multiple device driver support (RAID and LVM)`, che supporta la creazione di RAID-0, RAID-1, RAID-4, RAID-5, RAID-6, RAID-10 (questi ultimi sui kernel più recenti) e *linear*. È cioè possibile far costruire al kernel stesso degli array utilizzando qualunque tipo di dispositivo a blocchi. Dato che gli stessi RAID software sono a loro volta dispositivi a blocchi, è possibile riutilizzarli per metterli in RAID fra di loro, pertanto è possibile creare a mano un RAID-10 mettendo semplicemente in RAID-0 un precedente array software creato in RAID-1. Una volta costruiti gli array si potrà accedere ad essi con i file di dispositivo `/dev/mdN` dove `N` è un numero progressivo (in genere si parte da zero) associato a ciascun dispositivo RAID in fase di creazione.

A parte abilitare il supporto nel kernel (cui abbiamo accennato in sez. 5.2.3), per l'utilizzo del RAID software è necessario utilizzare anche gli opportuni programmi in user space. Fino al kernel 2.4 di questi esistevano due versioni, i `raidtools` e il programma `mdadm`. Con i kernel della serie 2.6 tutto viene gestito con `mdadm` che è un singolo programma che raccoglie al suo interno tutte le funzionalità che prima erano suddivise nei vari componenti dei `raidtools`. Questi oggi sono totalmente deprecati e non sono più disponibili in tutte le distribuzioni più recenti.

6.1.3 La gestione del RAID software

Per la creazione di un RAID software occorre anzitutto disporre di un opportuno numero di dischi da utilizzare come membri del RAID, questo significa almeno 2 per il RAID-1, almeno 3 per il RAID-5 ed almeno 4 per RAID-6 e RAID-10. Benché sia possibile utilizzare un intero disco come membro di un array, è molto più comune utilizzare delle singole partizioni (cosa che ad esempio nel caso del RAID-1 consente di riutilizzare direttamente anche il singolo disco). Questo consente anche, suddividendo i dischi in partizioni uguali, di creare più dispositivi RAID dallo stesso insieme di dischi, da utilizzare nella stessa maniera in cui si usano le diverse partizioni di un unico disco.

Si tenga presente però che non ha alcun senso usare due partizioni dello stesso disco come membri di un RAID; infatti in tal caso gli stessi dati verrebbero scritti su due parti diverse dello stesso disco e non si avrebbe così alcuna ridondanza. Inoltre dato che i dati di un RAID vengono scritti in contemporanea su tutti i membri dell'array una tale configurazione comporterebbe uno stress notevole del disco che dovrebbe eseguire accessi contemporanei su zone diverse, e quindi un grosso lavoro di spostamento delle testine; le conseguenze sarebbero un enorme degrado delle prestazioni ed un forte stress meccanico, con aumento della probabilità di rottura.

Una seconda considerazione deve essere fatta riguardo la configurazione hardware dei dischi. Si deve infatti tener presente che con il RAID software i dati vengono inviati in copia su ciascun

disco, questo significa ad esempio che se si hanno molti dischi veloci su un unico bus SCSI, si potrebbe raggiungere il limite di banda passante del bus e non sfruttare a pieno la velocità di trasferimento dei singoli dischi.

Un problema simile si ha quando si usano dischi IDE, in tal caso infatti se si montano due dischi sullo stesso canale IDE essendo l'accesso al bus conteso le prestazioni degradano dato che i dati possono essere inviati soltanto un disco alla volta; inoltre se uno dei dischi si rompe questo può comportare anche il blocco dell'intero canale, con il conseguente disastroso fallimento dell'intero array. Per questo motivo quando si usano dischi IDE questi devono sempre essere posti su canali IDE diversi. Questo comunque non si applica al Serial ATA, che non supporta il concetto di master/slave.

Si tenga presente infine che, a parte il caso del `linear`, per poter creare un RAID occorrerà che i dischi (o le singole partizioni) che andranno a far parte dell'array siano delle stesse dimensioni. In realtà questo non è strettamente necessario, in quanto se si usano dischi o partizioni di dimensioni diverse in fase di creazione del RAID verrà usata la più piccola di quelle disponibili, e l'unico inconveniente potrà essere quello dello spreco di un po' di spazio disco.

Le cose però possono diventare molto più complicate in caso di sostituzione di un disco: accade spesso infatti che le dimensioni dichiarate dai costruttori siano tutt'altro che precise, ed è comune trovarsi con due dischi che solo nominalmente sono della stessa dimensione, il problema è che se il nuovo disco è anche solo leggermente più piccolo della dimensione usata per il RAID con il vecchio, non è più possibile utilizzarlo. Per questo c'è chi usa sempre una partizione leggermente più piccola delle dimensioni reali del disco, per lasciarsi un margine per casi di questo tipo.

Come accennato in sez. 5.1.2 le partizioni utilizzate per il RAID software possono essere marcate come tali impostando un tipo specifico; i valori possibili sono due, "`fd`" che identifica le vecchie partizioni di tipo "*RAID autodetect*" e" `da`" che è il valore consigliato per le nuove partizioni (denominato "*non fs data*"). L'uso di "`fd`" infatti comporta da parte del kernel il tentativo di assemblare e far partire automaticamente il RAID software quando rileva una partizione di questo tipo,[4] cosa che non è detto sia sempre voluta, e che in certi casi, come quello in cui si sposta un disco su un'altra macchina senza cancellarlo, può creare danni notevoli se questo viene assemblato come parte di un array già attivo.

Per questo motivo questa funzionalità, creata per poter far partire automaticamente un RAID da montare come radice, è attualmente deprecata (anche se sempre disponibile) in favore dell'uso di un opportuno *ramdisk* di avvio (vedi sez. 5.2.4) contenente i programmi in grado di assemblare il RAID in modo opportuno, ad esempio facendo riferimento agli UUID associati a ciascun dispositivo RAID software.

Come già accennato a partire dai kernel della serie 2.6 tutte le operazioni di gestione del RAID software sono state demandate al programma `mdadm`; dato che detto programma è disponibile anche per i kernel della serie 2.4 non tratteremo affatto i vecchi `raidtools` il cui interesse ormai è esclusivamente storico.

Una delle caratteristiche principali di `mdadm`, che lo distingue nettamente dai `raidtools`, è quella di poter funzionare anche senza la presenza di un file di configurazione; la scelta deriva

[4] questa funzionalità è comunque disponibile solo se il supporto è compilato all'interno del kernel e solo con certe versioni del RAID software, ed in particolare con la versione 0.90 del cosiddetto *RAID superblock*, che al momento (Aprile 2009) è ancora il default.

dal fatto che quando la radice è su RAID il contenuto di /etc non è accessibile senza aver prima attivato il RAID stesso.

Il comando fa comunque riferimento al contenuto /etc/mdadm/mdadm.conf o, se questo non esiste, di /etc/mdadm.conf. In genere questo file viene utilizzato, oltre che per impostare una serie di valori di default, per mantenere la lista degli array presenti sulla propria macchina e viene sempre creato in fase di installazione. Un esempio del suo contenuto è quello riportato dal seguente estratto dalla versione presente su una Debian Lenny su cui sono stati creati tre array RAID-1:

mdadm.conf

```
# by default, scan all partitions (/proc/partitions) for MD superblocks.
# alternatively, specify devices to scan, using wildcards if desired.
DEVICE partitions

# auto-create devices with Debian standard permissions
CREATE owner=root group=disk mode=0660 auto=yes

# automatically tag new arrays as belonging to the local system
HOMEHOST <system>

# instruct the monitoring daemon where to send mail alerts
MAILADDR root

# definitions of existing MD arrays
ARRAY /dev/md0 level=raid1 num-devices=2 UUID=63492dc1:8eaa5c05:5fb94220:239df376
ARRAY /dev/md1 level=raid1 num-devices=2 UUID=fa83ab9c:8d008d80:ded268eb:c0534d8f
ARRAY /dev/md2 level=raid1 num-devices=2 UUID=e11003b2:7eda78aa:0e5f395c:9308b30d
```

Il formato di mdadm.conf è quello tipico in cui le righe vuote e quelle inizianti con il carattere "#" vengono ignorate; le configurazioni vengono specificate ponendo la parola chiave che identifica la direttiva all'inizio della riga, seguita poi da eventuali parametri separati da spazi o tabulazioni. Una riga che inizia con un carattere di spaziatura viene considerata una prosecuzione della riga precedente. Le principali direttive ed i loro parametri sono illustrate in tab. 6.1, le parole chiavi ivi riportate possono essere scritte per intero o con una abbreviazione di tre lettere. Al solito per l'elenco completo di direttive e parametri ed i dettagli sul loro significato si può fare riferimento alla pagina di manuale accessibile con man mdadm.conf.

Dato che tutti gli aspetti della gestione del RAID software vengono gestiti con mdadm, le operazioni del comando sono state suddivise su sette diverse modalità di utilizzo, riportate in tab. 6.2. Le prime cinque modalità sono da attivare esplicitamente tramite l'uso delle corrispondenti opzioni, mentre la modalità *Manage* è sottintesa se si passa come primo argomento prima di ogni opzione il file di dispositivo di un array o se la prima opzione è --add, --fail o --remove. Tutte le restanti possibilità fanno capo alla modalità *Misc*.

In generale nella gestione del RAID il primo utilizzo di mdadm deve essere effettuato in modalità *Create* (con l'opzione -C o --create) per creare l'array a partire dalla serie di dischi che ne fanno parte, a cui deve seguire il nome del file di dispositivo a cui l'array sarà associato. Si tenga presente che in questo caso creare significa predisporre su ciascuno dei membri di un array i dati che ne contengono le caratteristiche e le informazioni di gestione, che sono mantenuti nel cosiddetto *superblocco* (in inglese *superblock*); una piccola sezione di disco utilizzata per mantenere

Opzione	Significato
`ARRAY`	identifica un array, il primo parametro è il nome del file di dispositivo ad esso associato, seguito da una lista di parametri che identificano le caratteristiche dell'array; questi sono nella forma `chiave=valore` dove ciascuna chiave identifica la caratteristica indicata dal corrispondente valore, le principali chiavi sono: `uuid` che indica l'UUID associato all'array, `level` che indica il tipo di RAID, `num-devices` che indica il numero di componenti; per un elenco completo si consulti la pagina di manuale.
`DEVICE`	indica la lista dei dispositivi che possono contenere un componente di un array, questi possono essere indicati tramite i rispettivi file di dispositivo, usando anche i caratteri jolly del *filename globbing*, alternativamente si può usare la parola chiave `partitions` per indicare al comando di leggere ed utilizzare il contenuto di `/proc/partitions`.
`CREATE`	indica i permessi da assegnare al file di dispositivo di un array quando questo viene creato in modalità automatica (con l'opzione `-a` di `--create`); prevede l'uso di una serie di parametri nella forma `chiave=valore`, di nuovo si consulti la pagina di manuale per i dettagli, anche se i nomi delle chiavi, come `user`, `group` o `mode` sono piuttosto espliciti.
`MAILADDR`	indica l'indirizzo email, a cui inviare gli allarmi quando `mdadm` viene eseguito in modalità *Monitor*.
`HOMEHOST`	indica il nome della macchina usato come valore di default per l'opzione `--homehost` di `mdadm`; può essere un nome a dominio o la parola chiave `<system>` che la ottiene direttamente dal sistema (si assumono noti i concetti di nome a dominio e di nome della macchina che tratteremo comunque in sez. 7.4).

Tabella 6.1: Principali direttive di configurazione di `mdadm.conf`.

le informazioni relative all'array di cui esso viene a far parte.[5] Fra queste la più importante è il valore dell'UUID associato all'array, che è quello che poi può essere utilizzato in seguito per farlo ripartire (al riavvio della macchina o qualora lo si fosse fermato), e che compare fra i parametri della direttiva `ARRAY` di `mdadm.conf`.

Sempre nel superblocco viene registrato il nome della macchina su cui si è creato l'array, cosa che consente di evitare che questo venga attivato automaticamente su una macchina diversa quando si usano le opzioni per la partenza automatica dell'array. Il nome da usare viene impostato tramite la direttiva `HOMEHOST` di `mdadm.conf` il cui default, come mostrato nell'esempio precedente, è `<system>`, che indica di usare il nome stesso della macchina. Il valore può però anche essere specificato direttamente sulla riga di comando, soprassedendo quanto indicato con detta direttiva, come argomento per l'opzione `--homehost`, questa è una delle opzioni generiche che possono essere usate in diverse modalità, il suo compito è quello di indicare quale nome di macchina deve essere preso come riferimento nei confronti di quello memorizzato sul superblocco.

Si tenga presente che nella gestione del RAID software la modalità *Create* deve essere impiegata soltanto una volta, quando si costruisce l'array; in seguito infatti sarà sufficiente utilizzare le informazioni permanenti registrate nel superblocco per fare ripartire l'array riassemblando (come vedremo analizzando la modalità *Assemble*) i vari dischi che ne fanno parte. Si può però anche creare un array con la modalità *Build* in cui questo viene costruito senza registrare informazioni permanenti sui suoi componenti; in tal caso non esistono controlli che permettono di

[5] fino alla versione 0.90 questo era posto in coda al disco (o alla partizione), è stato poi spostato all'inizio e nell'ultima versione a 4K dall'inizio della partizione, è cura del RAID software evitare che lo spazio allocato per il *superblocco* venga sovrascritto dai dati.

Modalità	Opzione		Significato
Create	`--create`	`-C`	crea da zero un nuovo array di dischi, registrando sui singoli dischi le informazioni di gestione ad esso relative con l'installazione del cosiddetto *superblocco*.
Build	`--build`	`-B`	costruisce ed attiva un array ignorando eventuali informazioni di gestione del *superblocco*, pertanto è possibile costruire un array con dischi anche completamente scorrelati fra loro distruggendo completamente i dati precedentemente presenti, inoltre per riavviare l'array occorre ricordarsi esattamente i dischi usati, è pertanto da usare esclusivamente quando si sa esattamente cosa si sta facendo.
Assemble	`--assemble`	`-A`	assembla un array precedentemente creato con la registrazione del *superblocco* sui singoli dispositivi; viene utilizzato per far partire un dispositivo RAID all'avvio.
Grow	`--grow`	`-G`	consente di allargare o restringere un array, o di modificarne le caratteristiche. In pratica viene usato per modificare le dimensioni dei singoli componenti di un array o per cambiare il numero di dischi attivi per i RAID del livelli 1, 4, 5 e 6.
Monitor	`--monitor`	`-F`	denominato anche *Follow* (attivabile anche con `--follow`) consente di tenere sotto osservazione lo stato di integrità di uno o più array; ha senso solo per gli array che supportano la ridondanza.
Incr. Assembly	`--incremental`	`-I`	una variante di *Assemble* in cui si specificano i membri di un array uno alla volta invece che tutti insieme; viene in genere utilizzato per far partire un RAID quando questo è composto di dischi *hot-plug* che possono essere riconosciuti in sequenza.
Manage	–	–	è la modalità in cui si può operare sui singoli componenti di un array, con operazioni come l'aggiunta di nuovi dischi di riserva, la rimozione dall'array di dischi difettosi e simili.
Misc	–	–	è la modalità utilizzata per tutte le operazioni di varia natura che non rientrano nelle categorie precedenti.

Tabella 6.2: Modalità di operazione di `mdadm`.

evitare di assemblare un array con dati non coerenti, e in caso di errore si rischia ovviamente la perdita completa del contenuto dello stesso.

Una volta selezionata la modalità con `--create` si deve specificare come primo argomento il file di dispositivo che si vuole sia usato per accedere all'array. Si può però usare l'opzione `-a` per richiedere una allocazione dinamica dello stesso; in tal caso il dispositivo viene creato sotto `/dev/md/` con un numero progressivo anche se il classico `/dev/mdN` viene fornito come link simbolico al precedente, si deve poi specificare il tipo di RAID con l'opzione `-l` (o `--level`),[6] che richiede un successivo parametro che indichi il "*livello*". Questo deve essere specificato con una delle parole chiave `stripe` o `mirror`, sinonimi rispettivamente di RAID-0 e RAID-1, oppure specificando direttamente il valore del livello nella forma `raidN` o semplicemente con `N`, (dove `N` può assumere i valori 0, 1, 4, 5, 6 e 10).

L'altra opzione obbligatoria in fase di creazione di un array è `-n` (o `--raid-devices`) con cui si specifica il numero di dispositivi che ne fanno parte; all'opzione deve essere fatta seguire una lista di altrettanti file di dispositivo, si può però includere nella lista, per i livelli di RAID che

[6] si ricordi che in caso di opzioni estese `mdadm` richiede che il relativo valore venga specificato in forma di assegnazione.

supportano la ridondanza, la parola chiave `missing`,[7] che consente di creare un array senza la presenza di tutti i dischi dichiarati; ovviamente in questo caso l'array creato sarà degradato fin dall'inizio, ma si potranno aggiungere i dischi mancanti in seguito.

Se si vogliono impostare anche dei dischi di riserva il loro numero deve essere specificato con l'opzione `-x` (o `--spare-devices`) cui come nel caso precedente si deve far seguire l'elenco dei relativi file di dispositivo. In realtà si possono specificare le due opzioni ed i file di dispositivo in un ordine qualunque, ma il numero di file di dispositivo indicati (comprendendo fra questi eventuali `missing`) deve corrispondere alla somma dei due numeri.

Pertanto un possibile esempio di creazione con `mdadm` di un dispositivo RAID-1 a partire da due dischi IDE su cui si è usata una unica partizione potrebbe essere il seguente:

```
mdadm --create /dev/md0 --level=mirror --raid-devices 2 /dev/hda1 /dev/hdc1
```

mentre per la creazione di un RAID-5 a partire da tre dischi SCSI, con un disco IDE come spare, sempre con una unica partizione, si potrebbe usare il comando:

```
mdadm -C /dev/md1 -l 5 -n 3 /dev/sda1 /dev/sdb1 /dev/sdc1 -x 1 /dev/hda1
```

Una volta creato un array con `--create` questo viene immediatamente attivato, questo comporta anche, per i livelli che supportano la ridondanza, l'avvio della procedura di sincronizzazione dei dati dei vari dischi che lo compongono (e il conseguente inizio di una frenetica attività sui dischi). Lo stato degli array RAID presenti su una macchina può essere controllato tramite il file `/proc/mdstat`. Un esempio tipico di questo file, in una situazione ordinaria di utilizzo di un array, è il seguente:

```
gethen:~# cat /proc/mdstat
Personalities : [raid1]
read_ahead 1024 sectors
md0 : active raid1 hdc1[1] hda1[0]
      116214080 blocks [2/2] [UU]

unused devices: <none>
```

in cui viene mostrato lo stato su una macchina in cui è presente un solo array RAID-1 composto da due dischi, sincronizzato e attivo.

In una situazione in cui un array è stato appena creato, e pertanto il contenuto dei membri deve esser ancora sincronizzato, si avrà invece un risultato analogo al seguente:

```
alterra:~# cat /proc/mdstat
Personalities : [raid1]
md2 : active raid1 sda3[0] sdb3[1]
      7618816 blocks [2/2] [UU]
      [=========>..........]  resync = 45.0% (3433728/7618816) finish=63.2min speed=11018K/sec
```

dove entrambi i dischi di un RAID-1 sono membri attivi dell'array, ma è in corso la sincronizzazione dei loro contenuti.

[7] il numero di volte in cui si può farlo dipende ovviamente dal tipo di RAID scelto: con i RAID-4 e RAID-5 una sola volta, con il RAID-6 due volte e quante volte si vuole con il RAID-1, ovviamente sarà necessario comunque indicare, per ciascuna di queste modalità, un numero di dispositivi sufficiente (almeno due, tranne che per il RAID-1).

Infine quando per un qualche motivo uno dei dischi che fa parte di un array viene disattivato ed in seguito riattivato, il RAID dovrà essere ricostruito; in tal caso si potrà avere invece un risultato analogo al seguente:

```
urras:~# cat /proc/mdstat
Personalities : [raid1]
read_ahead 1024 sectors
md0 : active raid1 hdc1[0] hda1[1]
      5855552 blocks [2/2] [UU]

md1 : active raid1 hdc2[2] hda2[1]
      69336448 blocks [2/1] [_U]
      [>....................]  recovery =  0.8% (581632/69336448) finish=183.7min speed=6232K/sec
md2 : active raid1 hdc3[2] hda3[1]
      4490048 blocks [2/1] [_U]
        resync=DELAYED
```

dove viene mostrato lo stato di una macchina con tre array RAID-1 costituiti dalle prime tre partizioni di due dischi IDE.

In questo caso per un errore il secondo disco (`/dev/hdc`) era stato disattivato, ed una volta riattivato `/proc/mdstat` mostra la situazione in cui il primo array ha completato la risincronizzazione dei dischi, il secondo è attivo con un solo disco ma in fase di ricostruzione ed il terzo è attivo su un solo disco, in attesa di essere a sua volta sincronizzato. Quando più array coinvolgono gli stessi dischi la sincronizzazione viene sempre effettuata un array alla volta per evitare eccessivo sforzo sulla meccanica del disco che altrimenti dovrebbe spostare le testine da una partizione all'altra.

Il formato di `/proc/mdstat` prevede la presenza di una intestazione che riporta la liste dei tipi di RAID utilizzati, seguita da una diversa sezione per ciascun array presente. Nell'esempio precedente si ha "`Personalities : [raid1]`" dato che in quel caso veniva utilizzato solo il RAID-1, altrimenti sarebbero state presenti anche voci corrispondenti alle altre "*personalità*".

Ciascuna delle sezioni inizia con il nome del file di dispositivo ad esso associato, seguito dallo stato e dalla lista dei dispositivi membri dell'array indicati dai rispettivi file di dispositivo con fra parentesi quadra la posizione dello stesso nell'array; il numero non significa nulla di per sé, è solo un indice in una tabella, ma serve ad identificare la posizione del dispositivo nella rappresentazione dei dispositivi attivi mostrata nelle righe successive.

La riga seguente indica la dimensione, in blocchi, dell'array, seguita dall'indicazione fra parentesi quadre del numero di dispositivi che ne fan parte verso quelli attivi. Lo stato dei singoli viene rappresentato graficamente anche da una stringa in cui ogni dispositivo attivo viene rappresentato da una lettera "`U`", mentre i dispositivi non attivi sono rappresentati dal carattere "`_`". In caso di ricostruzione, come nell'esempio precedente, viene mostrato anche una indicazione semigrafica dello stato della procedura di ricostruzione con una stima del tempo mancante alla sua conclusione.

Quando un array non è attivo, che questo sia avvenuto per il risultato di una operazione esplicita di blocco,[8] o perché si è all'avvio del sistema, esso deve essere esplicitamente riattivato[9] assemblando tutti i dispositivi che ne fanno parte usando `mdadm` in modalità *Assemble*. Le

[8] ad esempio si può fermare un qualunque array, se questo non è in uso, con il comando `mdadm --stop /dev/md0`.

[9] si ricordi che come accennato l'attivazione automatica viene eseguita dal kernel solo per i dispositivi marcati come membri di un RAID software nella tabella delle partizioni, e solo con la specifica versione del *superblocco*.

modalità con cui si può effettuare questa operazione sono molteplici; la più diretta è quella in cui si passa come primo argomento il nome del dispositivo associato all'array e come argomenti successivi l'elenco dei dispositivi che ne fanno parte; ad esempio nel caso dell'array RAID-1 creato con il primo dei comandi illustrati in precedenza si potrà far ripartire l'array con il comando:

```
mdadm --assemble /dev/md0 /dev/hda1 /dev/hdc1
```

Alternativamente si possono riavviare in un colpo solo tutti gli array che sono stati creati utilizzando soltanto l'opzione `--scan`; questa è una opzione generica presente in diverse modalità che consente di determinare automaticamente gli argomenti riguardanti la composizione di un array mancanti sulla riga di comando, come i dispositivi che ne fan parte, il file di dispositivo ad esso associato o gli UUID dello stesso. Pertanto se si vogliono avviare tutti gli array creati in precedenza si potrà usare semplicemente il comando:

```
mdadm --assemble --scan
```

L'uso della sola opzione `--scan` comporta una serie di operazioni abbastanza complesse; come primo passo viene usato, se presente, `mdadm.conf`, ed assemblati gli array ivi dichiarati con le direttive `ARRAY`, altrimenti viene usato il contenuto di `/proc/partitions` per identificare la presenza di dispositivi che possono far parte di un array e se il nome della macchina a cui fanno riferimento corrisponde a quello della direttiva `HOMEHOST` questi vengono fatti partire automaticamente.

Infine siccome che non è detto che si voglia far partire sempre tutti gli array presenti, ad esempio se si vogliono effettuare dei controlli sui singoli componenti e non avviare un array, o farlo partire in modalità degradata, è possibile avviare un singolo array utilizzando l'UUID che viene associato a ciascun RAID quando questo viene creato. In tal caso si dovrà usare l'opzione `-u` (o `--uuid`) per specificare il valore dell'UUID e con `--scan` verranno identificati ed utilizzati automaticamente solo i dispositivi ad esso corrispondenti.

Una volta assemblato un array con `--assemble` questo in genere viene anche avviato e reso disponibile, ma soltanto se si sono specificati negli argomenti tutti i componenti che ne fanno parte o si è usato `--scan` che li inserisce automaticamente. Nel caso di RAID che supportano la ridondanza è possibile anche far partire un array in *modalità degradata*, escludendo ad esempio, fintanto che gli altri sono sufficienti a far funzionare l'array, alcuni dei suoi componenti; in tal caso però deve essere specificata l'opzione `--run`, altrimenti l'array sarà assemblato, ma non avviato.

Si tenga presente comunque che difficilmente, a parte il caso di problemi, capiterà di eseguire queste operazioni a mano; gli script di avvio usati da tutte le distribuzioni infatti si incaricano usualmente di avviare automaticamente tutti gli array di una macchina. Qualora però per un qualche motivo detto avvio fallisse, cosa che può accadere se ad esempio i dischi che ne fanno parte non sono stati ancora riconosciuti al momento in cui se ne richiede l'avvio, si potrà far ripartire un array utilizzando `mdadm` in questa modalità.

Nella gestione ordinaria, fintanto che non ci sono problemi, è assai raro dover usare `mdadm`. La situazione cambia quando un componente di un array fallisce, come nel caso di un disco che si rompe, per cui ci si troverà in presenza di un array degradato e si dovranno prendere gli opportuni provvedimenti. Il primo passo è accorgersi di queste situazioni e per questo motivo `mdadm` prevede una apposita modalità *Monitor*, che normalmente viene attivata direttamente all'avvio dalla gran parte delle distribuzioni. Ovviamente la modalità ha senso solo per i livelli

che supportano la ridondanza, in cui l'accesso ai dati continua a funzionare, negli altri casi il fallimento diventa immediatamente evidente.

Se usato in questa modalità `mdadm` resta attivo e continua a controllare lo stato degli array, registrando eventuali cambiamenti di stato ed inviando gli allarmi via email all'amministratore. In genere il comando viene fatto eseguire in *background* come demone, usando l'opzione `-f` (o `--daemonise`), direttamente dagli script di avvio, altrimenti resta attivo sulla console stampando i suoi messaggi sullo *standard output*. In questa modalità si può specificare la lista degli array da tenere sotto controllo con i relativi file di dispositivo, ma al solito l'opzione `--scan` consente di mettere sotto controllo in un colpo solo tutti quelli definiti in `mdadm.conf`.

Il programma rileva una serie di cambiamenti di stato, e genera altrettanti eventi, che, usando l'opzione `-y` (o `--syslog`), vengono registrati sul sistema del *syslog*. Inoltre per alcuni di questi eventi, quelli che indicano una situazione problematica, come il fallimento di un disco o il degrado di un array, è previsto l'invio automatico di una email di avvertimento all'indirizzo specificato nella direttiva `MAILADDR` di `mdadm.conf`, che può comunque essere soprasseduto con l'opzione `-m` (o `--mail`). Altre opzioni significative sono `-d` (o `--delay`), che consente di modificare l'intervallo di tempo in cui vengono effettuati i controlli e `-l` (o `--oneshot`) che consente di eseguire un controllo singolo; al solito per i dettagli, e per il significato e le priorità dei vari eventi, si consulti la pagina di manuale.

Una volta che si sia stati notificati di un problema sugli array, occorrerà provvedere alla sua soluzione; per far questo la modalità da usare è la *Manage*. Il caso classico è quello della rottura di un disco che dovrà essere sostituito nel più breve tempo possibile; il primo passo è rimuovere il disco rotto dall'array con l'opzione `-r` (o `--remove`), dopo di che si potrà spegnere la macchina (a meno di non avere dischi che supportano un cambiamento a caldo) e sostituire il disco. Al riavvio il nuovo disco potrà essere reinserito nell'array con `-a` (o `--add`). Si tenga presente che si possono rimuovere da un array soltanto i dischi non in uso, vale a dire o dischi con errori che sono stati marcati come *fail* o dischi *spare*. È anche possibile, con l'opzione `-f` (o `--fail`), forzare il fallimento di un disco funzionante, cosa che si può usare ad esempio qualora si voglia sostituire un disco anche se non è fisicamente rotto.

In tutti questi casi `mdadm` richiede che si indichi sempre per primo il file di dispositivo relativo all'array su cui si opera mentre i successivi devono fare riferimento ai dischi che ne fanno parte; è possibile infatti eseguire più operazioni su una sola riga di comando, anche se poco consigliabile vista la scarsa leggibilità che si ottiene in tal modo. Se allora nell'esempio di RAID 1 mostrato in precedenza fallisce il secondo disco IDE, si potrà provvedere alla sua sostituzione dopo averlo rimosso con il comando:

```
mdadm /dev/md0 --remove /dev/hdc1
```

ed una volta sostituito si potrà inserire il nuovo disco nell'array, (supponendo di non aver cambiato canale IDE), con:

```
mdadm /dev/md0 --add /dev/hdc1
```

Delle ulteriori funzionalità avanzate di gestione degli array sono state introdotte nello sviluppo del kernel 2.6 con la modalità *Grow* che consente di modificare le dimensioni degli array. Fino allora infatti tutto quello che si poteva fare era aggiungere dischi *spare*, operazione che aumenta la ridondanza, ma non lo spazio disponibile. In modalità *Grow* diventa invece possibile

far crescere lo spazio disponibile; ovviamente il funzionamento dipende dal tipo di RAID che si sta utilizzando.

Esistono due modi per aumentare le dimensioni di un array; nel caso di RAID nei livelli 4, 5 o 6 si può aggiungere un disco a quelli presenti, richiedendo che le informazioni vengano distribuite anche su di esso, così da aumentare lo spazio disponibile, come sarebbe stato se si fosse creato l'array fin dall'inizio inserendovi anche il disco aggiuntivo. La seconda modalità prevede invece la sostituzione dei dischi componenti un array con dischi di dimensioni maggiori, e può essere utilizzata anche con RAID di livello 1. Si tenga presente che in entrambi i casi quello che cambierà al termine delle operazioni di *crescita* sarà la dimensione del dispositivo RAID, non del filesystem in esso contenuto, che andrà opportunamente ridimensionato con i comandi che vedremo in sez. 6.3.2.

Nel primo caso la procedura prevede che prima si effettui l'aggiunta del disco e poi si richieda l'accrescimento dell'array specificandone il nuovo numero di dischi. In sostanza per aggiungere un nuovo disco ad un RAID 5 già composto da tre dischi, in modo da estenderne le dimensioni, si dovrà prima aggiungere il disco all'array con un comando analogo al seguente:[10]

```
mdadm /dev/md1 --add /dev/sdd1
```

e poi utilizzare `mdadm` in modalità *Grow* per accrescere l'array indicandogli il nuovo numero di componenti, con qualcosa come:

```
mdadm --grow /dev/md1 --raid-devices=4
```

Si tenga presente che questo tipo di operazione richiede che i dati vengano redistribuiti fra i vecchi ed il nuovo disco; questo procedimento è molto oneroso e può durare parecchie ore (fino al limite delle intere giornate). Inoltre all'inizio della procedura c'è un periodo critico in cui le informazioni trasferite non possono essere salvate in maniera sicura sull'array stesso. Per questo motivo, onde evitare che in caso di spegnimento accidentale l'array si trovi in uno stato inconsistente (e probabilmente irrecuperabile), è sempre opportuno usare l'opzione `-backup-file` per indicare un file su cui salvare dette informazioni, che ovviamente deve stare su un dispositivo diverso dall'array stesso.

Qualora invece si voglia operare un accrescimento dell'array attraverso il rimpiazzo dei suoi componenti con dischi di dimensione maggiore il primo passo è effettuare la sostituzione successiva di ciascuno di essi; per far questo prima occorre marcare un disco come difettoso con un comando analogo al seguente:

```
mdadm /dev/md0 --fail /dev/hda1
```

e poi rimuoverlo dal RAID con:

```
mdadm /dev/md0 --remove /dev/hda1
```

a questo punto si potrà sostituire il disco come nel precedente caso di rottura, utilizzandone un altro di dimensioni maggiori (che si sarà opportunamente partizionato), per poi riaggiungerlo al RAID con:

```
mdadm /dev/md0 --add /dev/hda1
```

[10]facendo ovviamente riferimento agli opportuni file di dispositivo relativi al proprio array.

Si dovrà poi ripetere l'operazione per tutti i dischi che compongono l'array, aspettando dopo ogni sostituzione che l'array si risincronizzi. Una volta fatto questo si avrà un array identico a quello di partenza, in cui però viene usata solo una parte dello spazio disponibile sui dispositivi componenti, il passo finale è quello di usare `mdadm` in modalità *Grow* per eseguire l'accrescimento dell'array usando l'opzione `-z` (o `--size`) per indicare la nuova dimensione con qualcosa del tipo:

```
mdadm --grow /dev/md0 --size=max
```

dove `max` è una parola riservata che richiede che la dimensione dell'intero array venga portata al massimo dello spazio utilizzabile nei dischi che ne fan parte.

Si ricordi comunque che l'accrescimento di un array non comporta il ridimensionamento del filesystem da esso contenuto. Inoltre si tenga presente che le operazioni di accrescimento degli array non aggiornano automaticamente il contenuto di `mdadm.conf`, che deve essere modificato manualmente (questo in realtà deve essere fatto soltanto nel primo dei due casi trattati, cambiando il numero dei componenti).

In questo possono essere di ausilio alcune delle funzionalità fornite da `mdadm` in modalità *Misc*, in particolare si può combinare l'opzione `-D` (o `--examine`) che usualmente richiede l'indicazione del dispositivo di un array per stamparne caratteristiche e componenti, con l'opzione `--scan`, così da ottenere un risultato del tipo:

```
gethen:~# mdadm --examine --scan
ARRAY /dev/md0 level=raid1 num-devices=2 UUID=79448936:36120176:1eaacd19:dfef9fa6
```

che fornisce uno scheletro del contenuto di `mdadm.conf` a partire dal contenuto degli array stessi.

Lo stesso risultato si ottiene anche con l'opzione `-E` (o `--query`) che invece, prendendo sempre come argomento un file di dispositivo, permette di vedere se questo a seconda dei casi è un array o fa parte di un array. Se ad essa si aggiungono le ulteriori opzioni `--detail` e `--scan` si riotterrà il risultato precedente.

Altre opzioni significative della modalità *Misc* sono `-S` (o `--stop`) che disattiva un array, che ovviamente non deve essere in uso, `-o` (o `--readonly`) che porta l'array in sola lettura, e `-w` (o `--readwrite`) che lo porta in lettura e scrittura. Di nuovo per i dettagli si consulti la pagina di manuale di `mdadm`.

Esistono infine una serie di opzioni generiche che si applicano a qualunque modalità di operazione; a parte `--scan` e `--homehost` che abbiamo già visto, la principale di queste è `-c` (o `--config`) che consente di indicare un file di configurazione alternativo rispetto a `mdadm.conf`; inoltre passando a `-c` il valore `partitions` si indica a `mdadm` di ignorare un eventuale contenuto del file di configurazione, con un comportamento equivalente a quello di avere la sola riga `DEVICE partitions` in `mdadm.conf`. Altre due opzioni generiche sono `-v` che aumenta la prolissità del comando e `-q` che la riduce.

6.2 Il sistema del *Logical Volume Manager*

Tratteremo in questa sezione il sistema del *Logical Volume Manager*, uno speciale supporto nel kernel che permette di organizzare in maniera flessibile lo spazio disco per superare le difficoltà di modificare le dimensioni di una partizione qualora il filesystem in essa contenuto non abbia più spazio sufficiente. L'uso del *Logical Volume Manager* permette cioè di evitare alla

radice tutta le problematiche, evidenziate in sez. 5.1.2, relative alle diverse possibili strategie di partizionamento.

6.2.1 Introduzione

Uno dei problemi che la struttura di un filesystem unix-like presenta è che quando si riempie un dispositivo montato su una directory non se ne può aumentare la dimensione direttamente semplicemente aggiungendo un altro disco perché questo andrebbe montato altrove. Pertanto in casi come questi si è spesso costretti a complesse operazioni di backup per sostituire il disco riempitosi o a creare link simbolici spostando su altri dischi il contenuto di alcune delle directory di quello troppo pieno.

Per superare alla radice questo problema è stato creato il sistema del *Logical Volume Manager*, grazie al quale è possibile creare un filesystem invece che su un disco singolo o su una partizione, all'interno di un "*volume logico*". Quest'ultimo poi non è altro che un dispositivo virtuale creato a partire da un insieme di *volumi fisici*, che vengono opportunamente *riuniti* dal kernel in modo da presentarsi come un unico disco.

Il vantaggio è che i volumi logici possono essere ridimensionati a piacere senza necessità di toccare le partizioni o i dischi che stanno al di sotto, per cui usando i programmi per il ridimensionamento dei filesystem si possono allargare o restringere questi ultimi in maniera molto comoda, e con certi filesystem alcune operazioni possono essere eseguite a "*caldo*", con il filesystem montato. In questo modo se lo spazio disco su un volume logico si esaurisce basterà allargarlo, e se lo spazio disco sottostante non è sufficiente basterà comprare un nuovo disco ed agganciarlo al volume logico, per espanderne le dimensioni senza doversi preoccupare di ripartizionare, montare, fare link e spostare contenuti da un disco ad un altro.

Lo schema del funzionamento del *Logical Volume Manager* (da qui in breve LVM) è riportato in fig. 6.1. Il sistema consiste nell'introdurre un livello intermedio, quello del *gruppo di volumi* (indicato in figura con la notazione VG, da *volume group*), che viene a costituire l'interfaccia fra i volumi logici (indicati con LV da *logical volume*), che da esso si estraggono, ed i volumi fisici (indicati con PV, da *physical volume*), che lo vanno a costituire. In sostanza il kernel si cura di suddividere un disco fisico in tante piccole sezioni (dette *physical extent*, e indicate in figura con PE) che poi vengono rimappate[11] nei corrispondenti *logical extent* (indicati con LE) che vanno a costituire un volume logico.

Una delle funzionalità più interessanti di LVM è quella di poter utilizzare il sistema per poter creare degli *snapshot*, cioè poter creare una copia dello stato di un volume logico ad un certo istante utilizzando lo spazio libero nel *gruppo di volumi* di cui questo fa parte. Tutto ciò viene realizzato allocando dello spazio libero per contenere tutte le modifiche eseguite al volume originario. La versione corrente del filesystem sul volume logico originale registrerà tutte le modifiche nel nuovo spazio allocato, mentre il volume creato come *snapshot* conterrà i dati così come erano al momento della creazione.

Questo è molto comodo per poter eseguire dei backup per volumi di grosse dimensioni senza doversi preoccupare del fatto che durante l'esecuzione del backup i dati vengano modificati; basta assicurarsi che nel breve tempo necessario a creare lo *snapshot* non ci siano attività, dopo di che si potrà eseguire il backup con tutta calma su di esso mentre le operazioni continuano

[11] da cui il nome di *device mapper* che il sistema ha assunto a partire dal kernel 2.6.x, dove è stato riscritto da zero per migliorare stabilità e prestazioni.

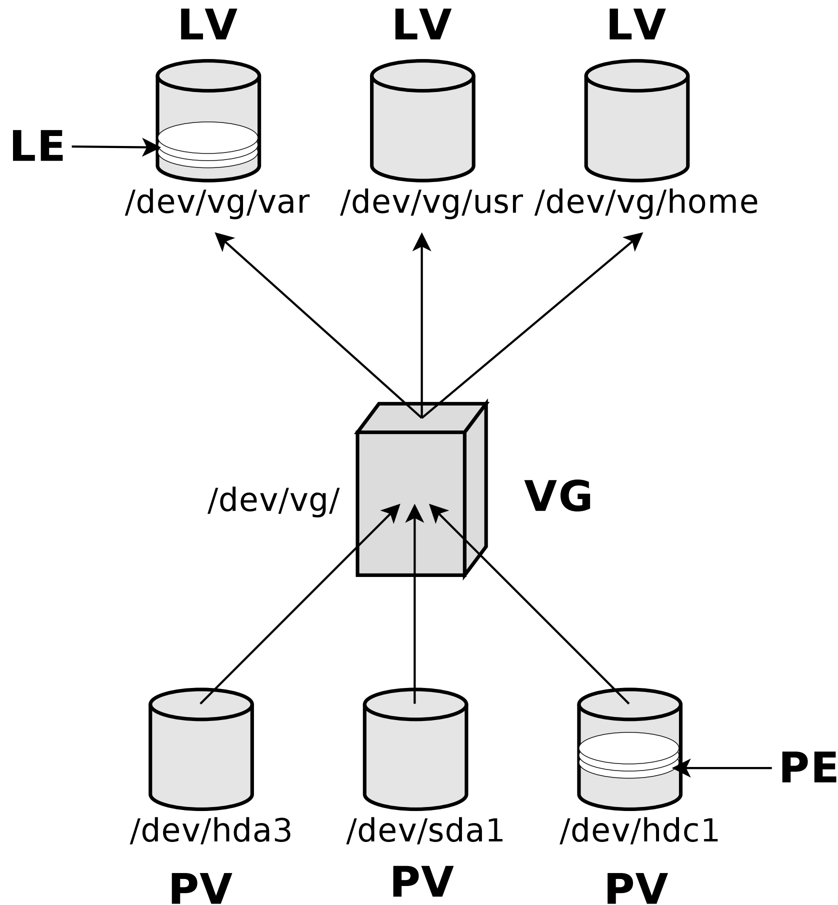


Figura 6.1: Strutturazione del *Logical Volume Manager*.

regolarmente sul volume originale. Una volta completato il backup si potrà rimuovere il volume logico usato per lo *snapshot* ed il volume logico originario manterrà in maniera trasparente tutte le modifiche eseguite nel frattempo; l'unico aspetto critico è quello di assicurarsi di avere allocato per lo *snapshot* spazio sufficiente a contenere tutte le modifiche, se infatti questo dovesse esaurirsi il volume logico originale diverrebbe inusabile.

Per poter utilizzare LVM occorre abilitare il relativo supporto nel kernel (vedi sez. 5.2.3), ed avere installato gli opportuni programmi in *user-space*. Nel passaggio dal kernel 2.4.x al 2.6.x il sistema è stato completamente riscritto, pertanto si potranno avere delle differenze a seconda che si usi la prima versione (detta anche LVM1) o la seconda (detta LVM2). In ogni caso la nuova versione è in grado di riconoscere la presenza di volumi logici creati con la versione precedente (ormai in completo disuso) e di riutilizzarli senza problemi, ed ha in genere prestazioni nettamente superiori.

6.2.2 La gestione dei *volumi fisici*

Il primo passo nell'uso di LVM è quello della inizializzazione dei volumi fisici; per questi può essere usato un qualunque dispositivo a blocchi. L'elenco dei dispositivi accettati può essere limitato con la direttiva `filter` nel file di configurazione `/etc/lvm/lvm.conf`, che non prevede alcuna esclusione. Il caso più comune è quello dell'uso di una partizione di un disco, che è opportuno marcare (vedi sez. 5.1.2) con il codice `0x8E` per indicare che verrà utilizzata come volume fisico per LVM. Una scelta altrettanto comune è quella di usare un dispositivo RAID software (nel qual caso non serve nessun altro accorgimento).

Ogni dispositivo che si intende usare come volume fisico deve essere inizializzato con il comando `pvcreate` così come si farebbe per un filesystem; il comando prende come argomento il file di dispositivo che identifica il volume fisico sul quale creare l'infrastruttura per mantenere le informazioni relative a LVM, e se ne possono specificare anche più di uno per una inizializzazione multipla. A meno di non impostare una dimensione inferiore con l'opzione `--setphysicalvolumesize` il volume occuperà l'intero spazio disponibile su dispositivo.

Oltre che ad una partizione il comando `pvcreate` può essere applicato anche ad un intero disco, nel qual caso però occorre prima cancellare la tabella delle partizioni, perché se questa viene rilevata il comando segnala un errore. Questo si può fare facilmente con un comando di cancellazione come:

```
dd if=/dev/zero of=/dev/hdX bs=512 count=1
```

Il comando prende come opzioni specifiche `-u`, che permette di specificare uno UUID a mano invece di farlo generare automaticamente, e `-f` che forza la creazione del volume senza richiedere conferma, distruggendo all'instante eventuali dati presenti. Le altre opzioni riguardano aspetti specifici della gestione interna ed al solito rimandiamo alla pagina di manuale, ma in genere non c'è bisogno di usare niente altro che il comando applicato al dispositivo, ad esempio con un comando del tipo:

```
hain:~# pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created
```

si inizializzerà la prima partizione del secondo disco SCSI come volume fisico per LVM.

Con il comando `pvscan`, si può eseguire una scansione dei dischi presenti per rilevare al loro interno la presenza di volumi fisici usati da LVM, il comando si invoca senza argomenti, e si avrà un risultato del tipo:

```
anarres:~# pvscan
  PV /dev/sda3   VG vg   lvm2 [132.23 GB / 85.73 GB free]
  PV /dev/md0    VG vg   lvm2 [931.00 GB / 737.57 GB free]
  Total: 2 [1.04 TB] / in use: 2 [1.04 TB] / in no VG: 0 [0   ]
```

che mostra la presenza di due volumi fisici associati ad un gruppo di volumi e parzialmente utilizzati.

Le opzioni del comando controllano sostanzialmente l'informazione che viene stampata a video, con `-s` si riduce la stessa al minimo, con `-u` si fanno stampare gli UUID presenti, con `-n` si stampano solo i volumi fisici che non appartengono ad un volume logico, per le altre opzioni si faccia riferimento alla pagina di manuale.

Un secondo comando diagnostico è `pvdisplay` che stampa a video le caratteristiche di uno o più volumi fisici, specificati come argomenti. Se non si specifica nulla il comando stampa i dati relativi a tutti i volumi fisici presenti, altrimenti se ne può indicare uno (o più) passando come argomenti i relativi file di dispositivo; nel nostro caso visto che abbiamo appena creato un volume fisico su `/dev/sdb1` potremo controllarne le caratteristiche con:

```
monk:~# pvdisplay /dev/sdb1
  "/dev/sdb1" is a new physical volume of "1,91 GiB"
  --- NEW Physical volume ---
  PV Name               /dev/sdb1
  VG Name
  PV Size               1,91 GiB
  Allocatable           NO
  PE Size               0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               DqBTm8-F9Tk-Eczy-4Slb-RTwl-8Vs0-54j61P
```

mentre nel caso lo si usi su una macchina in cui è presente un volume fisico in uso, avremo invece qualcosa del tipo:

```
hain:~# pvdisplay
  --- Physical volume ---
  PV Name               /dev/md2
  VG Name               vg
  PV Size               462,96 GB / not usable 2,31 MB
  Allocatable           yes
  PE Size (KByte)       4096
  Total PE              118518
  Free PE               10763
  Allocated PE          107755
  PV UUID               VO7DXX-4hMk-CDuE-PykW-y8bF-q7h0-pWCtHo
```

Anche per questo comando le opzioni servono a controllare l'informazione riportata; le principali sono `-s`, che fa stampare solo la dimensione totale del volume e `-c` che produce un output compatto in cui i vari dati relativi al volume sono separati da un carattere "`:`" così da poterlo usare facilmente per fare scansioni sulle loro proprietà all'interno di script. A queste si aggiunge l'opzione `-m` che consente di ottenere una mappa completa delle corrispondenze nelle allocazioni fra *physical extent* e *logical extent*. Al solito per tutti i dettagli si può fare riferimento alla pagina di manuale.

Un comando alternativo a `pvdisplay`, disponibile a partire dalla versione 2 di LVM, è `pvs`, che consente un controllo molto più dettagliato della stampa in colonne dei dati. Se invocato senza argomenti il comando stampa un riassunto in forma tabellare dei dati dei dischi, ad esempio:

```
anarres:~# pvs
  PV         VG   Fmt  Attr PSize   PFree
  /dev/md0   vg   lvm2 a-   931.00G 737.57G
  /dev/sda3  vg   lvm2 a-   132.23G  85.73G
```

Il comando al solito supporta l'indicazione dei dispositivi da analizzare, da passare come argomenti. Ma la vera potenza del comando è la possibilità di adattare la stampa delle informazioni sulla base dei parametri passati all'opzione `-o`, che consentono di ottenere i dettagli

sulle allocazioni dei *physical extent* ed i volumi logici cui questi sono associati; per i dettagli si rimanda di nuovo alla pagina di manuale.

Qualora si cambino le dimensioni di una partizione o di un dispositivo (ad esempio un RAID) che contiene un volume fisico, è possibile modificare anche le dimensioni di quest'ultimo con il comando `pvresize`, che prende come argomenti una lista di dispositivi su cui si sono installati dei volumi fisici. Il comando può operare su volumi fisici che sono già parte di un gruppo di volumi e che ospitano dati relativi a volumi logici; se dato senza opzioni i volumi fisici vengono ridimensionati automaticamente alla dimensione corrente del dispositivo sottostante.

Qualora sia necessario ridurre la dimensione di un volume fisico (ad esempio se è necessario restringere una partizione) si deve invece usare la speciale opzione `--setphysicalvolumesize` che richiede come parametro l'indicazione della dimensione da dare al volume fisico. Ovviamente qualora si debba ridurre la dimensione di una partizione su cui è presente un volume fisico è necessario eseguire prima questa operazione.

Infine un comando estremamente utile è `pvmove` che permette di spostare tutti i dati mantenuti su un certo volume fisico su di un altro, così da poterlo rimuovere dal gruppo di volumi di cui fa parte. Il comando prende come argomenti il volume fisico sorgente da cui rimuovere i dati e quello di destinazione su cui inviarli; ovviamente la destinazione deve avere spazio libero sufficiente a contenerli. Si può anche evitare di specificare un volume fisico di destinazione, nel qual caso lo spazio sarà redistribuito automaticamente sui volumi appartenenti allo stesso gruppo.

Il comando ha tre opzioni principali, la prima è `-i` che fa stampare una percentuale di completamento delle operazioni tutte le volte che è trascorso il numero di secondi passato come parametro; con `-n` invece si può restringere lo spostamento ai dati relativi al solo volume logico il cui nome si è passato come parametro. Infine con `--abort` si può abortire un precedente tentativo di spostamento non completato; se infatti il comando viene interrotto durante l'esecuzione non c'è nessun problema, e basterà reinvocarlo senza parametri per fare ripartire lo spostamento dal punto un cui era stato interrotto, a meno appunto di invocare `pvmove --abort`.

6.2.3 La gestione dei *gruppi di volumi*

Una volta inizializzati tutti i volumi fisici necessari il secondo passo è combinarli in un *gruppo di volumi*, per questo si deve usare il comando `vgcreate` che crea un nuovo gruppo di volumi assegnandogli il nome specificato come primo argomento, ed inserendoci tutti i volumi fisici specificati negli argomenti successivi. Perché il comando abbia successo tutti i volumi fisici devono essere stati preventivamente inizializzati con `pvcreate`. Una volta creato un gruppo di volumi con nome `vgname` si avrà a disposizione una corrispondente directory `/dev/vgname/` in cui si troveranno tutti i file (in particolare i file relativi ai volumi logici, che tratteremo in sez. 6.2.4).

L'opzione `-s` permette di definire le dimensioni dei *physical extent* in cui saranno divisi i volumi fisici, da passare come argomento numerico seguito dalle estensioni standard per indicare kilo, mega e gigabyte. La dimensione standard è di 4Mb, ed i valori possono spaziare da 4kb a 16Gb in potenze di due, si tenga presente che con la vecchia versione di LVM1 un volume logico può mantenere un massimo di 64k *physical extent* che comporta una dimensione massima di 256Gb, la nuova versione non ha un limite, ma un numero eccessivo di *physical extent* tende a peggiorare notevolmente le prestazioni, pertanto se si ha a disposizione molto spazio disco conviene usare delle dimensioni più ampie per i *physical extent*.

L'opzione `-l` permette di definire un numero massimo di volumi logici che possono essere estratti dal gruppo; il valore massimo assoluto era di 255 con LVM1, con LVM2 questa restrizione non sussiste più ed il default è 0 che indica che non c'è nessun limite. Con `-p` invece si imposta il numero massimo di volumi fisici che possono essere inclusi, ed anche in questo caso si aveva un massimo di 255 per LVM1 e nessun limite per LVM2. Con `-u` si può impostare manualmente un UUID. Per le ulteriori opzioni si può fare riferimento alla pagina di manuale, un elenco di opzioni comuni a buona parte dei comandi che operano con i gruppi ed i volumi logici si trova in tab. 6.3, l'elenco completo invece è riportato nella pagina di manuale dedicata a LVM, accessibile con `man lvm`, ma normalmente sarà sufficiente un comando come:

```
monk:~# vgcreate vg /dev/hda3
  Volume group "vg" successfully created
```

che crea il nuovo gruppo di volumi `vg` con il volume fisico `/dev/hda3` precedentemente inizializzato.

Opzione	**Significato**
`-v`	imposta il livello di prolissità, può essere ripetuto fino a tre volte per aumentare le informazioni stampate a video.
`-d`	importa il livello di debug, può essere ripetuto fino a sei volte per aumentare le informazioni stampate a video.
`-t`	esegue il comando in modalità di test, i dati non vengono modificati.
`-h`	stampa un messaggio di aiuto.
`-M`	imposta il formato dei meta-dati, prende come valore `lvm1` o `lvm2` (o semplicemente il numero), questa opzione è disponibile solo su LVM2.
`-A`	esegue automaticamente il backup dei meta-dati quando essi vengono cambiati, prende come valore `y` o `n`, (il default è il primo e non è il caso di cambiarlo).

Tabella 6.3: Principali opzioni comuni dei comandi di gestione di LVM.

Una volta creato un gruppo di volumi questo può essere espanso aggiungendovi altri volumi fisici con `vgextend`, che di nuovo prende come primo argomento il nome del gruppo, che stavolta deve esistere, seguito dai volumi fisici che si vogliono aggiungere, al solito specificati tramite il relativo file di dispositivo. Per i dettagli sulle opzioni (alcune delle quali sono riportate in tab. 6.3) si può fare riferimento alla pagina di manuale.

Il lavoro opposto, quello di rimuovere un volume da un gruppo, viene invece eseguito da `vgreduce` che al solito prende come primo argomento il nome del gruppo, seguito dai volumi che si vogliono rimuovere. Ovviamente perché il comando abbia successo tutti i volumi che si vogliono rimuovere dal gruppo non devono essere utilizzati; con l'opzione `-a` non è necessario specificare quali volumi rimuovere, dato che in tal caso si rimuoveranno tutti quelli non utilizzati. Al solito per i dettagli sul comando e le altre opzioni supportate si faccia riferimento alla pagina di manuale.

La creazione di un gruppo di volumi è comunque solo un primo passo, prima di poterlo utilizzare infatti questo deve essere *attivato* in modo da poter essere utilizzato. Per questo si usa in genere il comando `vgchange` il cui scopo è quello di modificare le proprietà di un gruppo di volumi. Il comando al solito prende come argomento il nome del gruppo di volumi, ma se non se ne specifica uno si applica a tutti quelli presenti.

L'opzione principale è `-a` che è quella che permette di attivare o disattivare il gruppo, rispettivamente con i parametri "`y`" e "`n`"; in genere gli script di avvio di LVM eseguono sempre un comando del tipo `vgchange -a y`. Le altre opzioni sono utilizzate principalmente per cambiare le caratteristiche del gruppo di volumi rispetto ai valori impostati in fase di creazione: `-l` modifica il numero massimo di volumi logici, passato come parametro (deve essere eseguita su un gruppo inattivo), `-x` attiva o disattiva (sempre con i parametri "`y`" e "`n`") la possibilità di estendere o ridurre il gruppo togliendo o aggiungendo volumi fisici, `-u` consente di modificare l'UUID associato al gruppo di volumi.

Un secondo comando usato all'avvio del sistema, ma solo per LVM1, in LVM2 la cosa è automatica, è `vgscan` che esegue una scansione dei dischi presenti per rilevare la presenza di gruppi di volumi (analogo a quanto fa `pvscan` per i volumi fisici), in genere lo si lancia prima di `vgchange`, e l'unica opzione significativa (per le altre si consulti la pagina di manuale) è `--mknodes` che crea in `/dev` i file speciali necessari all'uso dei dispositivi. Un esempio è:

```
monk:~# vgscan
  Reading all physical volumes.  This may take a while...
  Found volume group "vg" using metadata type lvm2
```

Esiste poi un comando analogo di `pvdisplay` da usare per i gruppi di volumi; con `vgdisplay` si possono elencare le caratteristiche del gruppo passato come argomento, o di tutti quelli presenti nel sistema se non si specifica nulla. Le opzioni sono le stesse già viste in sez. 6.2.2 per `pvdisplay`; un esempio di questo comando è:

```
monk:~# vgdisplay
--- Volume group ---
  VG Name               vg
  System ID
  Format                lvm2
  Metadata Areas        1
  Metadata Sequence No  1
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                0
  Open LV               0
  Max PV                0
  Cur PV                1
  Act PV                1
  VG Size               185,38 GB
  PE Size               4,00 MB
  Total PE              47457
  Alloc PE / Size       0 / 0
  Free  PE / Size       47457 / 185,38 GB
  VG UUID               njlk5d-xvsQ-pXcI-sEnK-zqII-YzzN-pNoI0Y
```

Quando si vogliono riunire in uno solo due gruppi di volumi si può utilizzare il comando `vgmerge`, questo prende come primo argomento il gruppo di destinazione, che diventerà l'unione dei due, e come secondo il gruppo da inglobare nel primo, che deve essere disattivo. Il comando funziona soltanto se i due volumi sono stati creati con parametri compatibili (stessa versione dei meta-dati e stessa dimensione dei *physical extent*) e se l'unione dei due rispetta i limiti presenti sul gruppo di destinazione. Per le opzioni (che sono sostanzialmente quelle comuni di tab. 6.3) e per gli altri dettagli si faccia riferimento alla pagina di manuale.

Se invece si vuole spezzare in due un gruppo di volumi si può usare il comando `vgsplit`, questo prende come primo argomento il nome del gruppo da cui si parte e come secondo quello del nuovo gruppo che si vuole creare, seguito dalla lista dei volumi fisici da inserire in quest'ultimo. Dato che i volumi logici non possono essere suddivisi fra gruppi diversi occorrerà che ciascun volume logico esistente sia compreso interamente nei volumi fisici che restano in uno dei due gruppi. Al solito per le opzioni (sono solo quelle comuni) ed i dettagli si può fare riferimento alla pagina di manuale.

6.2.4 La gestione dei *volumi logici*

Una volta creato un *gruppo di volumi* sarà possibile estrarre da esso lo spazio necessario a creare un *volume logico*. Il comando principale per eseguire questo compito è `lvcreate` che crea il volume logico allocando i *logical extent* necessari dalla riserva di *physical extent* disponibili.

Il comando richiede come argomento obbligatorio il nome del gruppo di volumi da utilizzare, inoltre deve essere sempre specificata la dimensione del volume logico che si va a creare; questo può essere fatto in due modi, o con l'opzione `-l`, seguita dal numero di *logical extent* da utilizzare,[12] o con l'opzione `-L` seguita dalla dimensione in byte; l'opzione supporta anche la specificazione di altre unità di misura come kilobyte, megabyte, gigabyte o terabyte se si aggiunge al numero passato come parametro l'iniziale (va bene sia maiuscola che minuscola) della misura utilizzata.

Se non si specifica altro il comando crea un volume logico con un nome assegnato automaticamente al valore `lvolN` dove `N` è il numero progressivo usato internamente dal sistema; si può però specificare un nome qualunque con l'opzione `-n lvname`; in questo caso si potrà accedere al nuovo volume logico usando come file di dispositivo `/dev/vgname/lvname`. Nei kernel della serie 2.6, dato che LVM è stato riscritto appoggiandosi direttamente al sistema del *device mapper* che si cura di effettuare la mappatura fra *physical extent* e *logical extent*, si potranno trovare i volumi logici anche direttamente sotto la directory `/dev/mapper`, accessibili direttamente con un nome del tipo `vgname-lvname`. Un esempio dell'uso più comune di questo comando è il seguente:

```
monk:~# lvcreate -L 50G -n data vg
  Logical volume "data" created
```

Una delle funzionalità di LVM è che permette di suddividere i dati su più volumi fisici in maniera analoga al RAID-0, definendo con l'opzione `-i` un numero di *stripes* su cui distribuire i dati. Valgono in questo caso le stesse considerazioni viste in sez. 6.1.1 per il RAID-0, in particolare si avrà un enorme degrado di prestazioni qualora si usassero partizioni diverse dello stesso disco come volumi fisici su cui distribuire le *stripes*. Se si utilizza questa funzionalità si può anche specificare la dimensione delle *stripes* con l'opzione `-I`.

Se invece di un normale volume logico si vuole creare uno *snapshot* occorre usare modalità completamente diversa di invocazione del comando. In tal caso occorre usare l'opzione `-s` e specificare un nome con l'opzione `-n`; inoltre gli argomenti obbligatori diventano due: il nome del volume logico da usare come *snapshot* e quello del volume logico cui fare da *snapshot*. Un elenco delle opzioni principali del comando `lvcreate` è riportato in tab. 6.4, il comando utilizza anche molte delle opzioni generali accennate in tab. 6.3; per i dettagli al solito si può fare riferimento alla pagina di manuale.

[12] le dimensioni sono le stesse dei *physical extent* sottostanti, definite in fase di creazione del gruppo.

Opzione	Significato
`-i`	indica di utilizzare il numero di *stripe* passato come parametro per distribuire i dati sui volumi fisici sottostanti.
`-I`	imposta la dimensione di una *stripe* nella distribuzione dei dati su più volumi fisici (da specificare con un numero `n` da 2 a 9 per indicare 2^n kilobyte).
`-l`	imposta la dimensione di un volume logico per numero (passato come parametro) di *logical extent*.
`-L`	imposta la dimensione di un volume logico specificata in byte, kilobyte, megabyte, gigabyte o terabyte.
`-n`	imposta il nome del volume logico, passato come parametro.
`-p`	importa i permessi sul volume logico, prende il valore `r` per indicare sola lettura o `rw` per indicare lettura e scrittura.
`-s`	indica la richiesta di creazione di un volume di *snapshot*.

Tabella 6.4: Principali opzioni del comando `lvcreate`.

Come per i volumi fisici ed i gruppi di volumi sono presenti due comandi, `lvscan` e `lvdisplay`, che possono essere usati rispettivamente per eseguire una scansione dei volumi logici presenti e per ottenere un sommario delle relative proprietà. Anche le opzioni sono analoghe e non staremo a ripeterle.

La comodità di LVM consiste nel fatto che una volta che si è creato un volume logico si potrà accedere a quest'ultimo con il relativo file di dispositivo e trattarlo come un qualunque disco; il vantaggio è che qualora si avesse la necessità di ampliarne le dimensioni si potrà ricorrere al comando `lvextend`, dopo di che non resterà che ridimensionare il filesystem presente sul volume logico (vedi sez. 6.3.2) per avere a disposizione lo spazio aggiunto.

Il comando richiede obbligatoriamente come argomento il volume da estendere e la dimensione finale dello stesso che deve essere specificata o con l'opzione `-l` in numero di *logical extent* o direttamente con l'opzione `-L` (con la stessa convenzione di `lvcreate`). Si possono però usare le stesse opzioni per specificare la dimensione dell'estensione al posto di quella totale, apponendo al valore specificato il carattere "`+`".

Il comando supporta anche l'uso delle *stripe* (con le stesse opzioni di tab. 6.4) ed inoltre si può specificare come argomenti ulteriori una lista di volumi fisici sui quali si vuole che sia compiuta l'estensione del volume logico. Il comando permette anche di estendere le dimensioni dei volumi di *snapshot*.

Se invece di aumentare le dimensioni di un volume logico le si vogliono ridurre si deve usare `lvreduce`. Il comando prende gli stessi argomenti ed opzioni di `lvextend` (esclusi quelli relativi alle *stripe*), solo che in questo caso la dimensione può essere specificata apponendovi un segno "`-`" per richiedere una riduzione. Si tenga presente che i dati presenti nella parte tolta con la riduzione vengono persi, pertanto è fondamentale ridurre le dimensioni di un eventuale filesystem presente sul volume logico *prima* di eseguirne una riduzione.

Per completare i comandi di ridimensionamento si deve citare il generico `lvresize` che unisce le funzionalità dei precedenti `lvreduce` e `lvextend` permettendo ridimensionamenti in qualunque direzione (sia crescenti che decrescenti). Di nuovo le opzioni sono le stesse, e stavolta sono permessi per le dimensioni sia il segno "`+`" che quello "`-`".

Qualora un volume logico non sia più utilizzato lo si può eliminare usando `lvremove`, il comando prende come argomento uno o più volumi da rimuovere, l'unica opzione specifica supportata

è `-f` che permette di sopprimere la richiesta di conferma della rimozione dallo standard input.

6.3 Gestione avanzata di dischi e filesystem

Si sono raccolti in questa sezione una serie di argomenti avanzati riguardo la gestione dei dischi e dei filesystem, come l'impostazione dei parametri di basso livello dei dischi, il ridimensionamento di filesystem e partizioni.

6.3.1 La gestione dei parametri dei dischi

Finora abbiamo sempre trattato la gestione dei dischi facendo riferimento alle operazioni da eseguire sui dati in essi contenuti all'interno di filesystem e partizioni; non abbiamo trattato affatto le operazioni di basso livello con cui il sistema può andare a modificare le caratteristiche e le modalità con si accede ai dischi, effettuate direttamente attraverso l'interfaccia con l'hardware degli stessi.

Tradizionalmente il programma per eseguire operazioni di basso livello sui dischi è `hdparm`, nato per impostare le proprietà dell'accesso ai dischi e le loro modalità di funzionamento attraverso l'interfaccia IDE/ATA, che a lungo è stata l'interfaccia predominante per l'uso dei dischi su Linux, quando i dischi SCSI, per costo e complessità, erano usati soltanto da una piccola percentuale di utenti. Il comando comunque è in grado di funzionare con le nuove interfacce *serial ATA* supportate tramite il sistema di *libata* del kernel e sui dischi USB che supportano la traduzione dei comandi ATA su SCSI (il cosiddetto SAT, *SCSI-ATA Command Translation*).

Il comando prende come argomento il file di dispositivo del disco (o altro dispositivo IDE) su cui si vuole operare, e se non si specifica altro stampa i valori di una serie di parametri relativi al dispositivo stesso (l'esecuzione senza opzioni è equivalente all'esecuzione con le opzioni `-acdgkmur`) più o meno completa a seconda della presenza o meno degli stessi o della capacità di accedere alla relativa informazione. Il comando prevede una lunga serie di opzioni di varia natura che consentono di effettuare impostazioni sulle funzionalità dei dischi o del bus o di ottenere informazioni dettagliate sugli stessi.

Un sottoinsieme di queste opzioni possono essere utilizzate sia per ottenere che per modificare le impostazioni correnti, a seconda che si specifichi soltanto l'opzione, nel qual caso il valore corrente sarà letto e stampato, o ad essa si faccia seguire un valore numerico, il cui valore dipendente dalla singola funzionalità, verrà usato come nuova impostazione.

Ad esempio con l'opzione `-a` è possibile modificare il numero di settori in *read ahead*, cioè letti in anticipo dal filesystem rispetto a quelli indicati da una operazione di lettura; in questo modo è possibile migliorare le prestazioni per la lettura di file di grandi dimensioni, eseguendo una *lettura preventiva* del contenuto successivo. Si tenga comunque presente che molti dischi hanno una funzione simile implementata a livello hardware, questa è realizzata a livello di sistema operativo e va eventualmente ad aggiungersi alla precedente.

Una seconda opzione importante è `-d` che consente di attivare o disattivare l'accesso in DMA al disco, che permette di aumentare notevolmente le prestazioni; un valore 1 abilita il DMA, ed un valore 0 lo disabilita. In genere questa funzionalità viene attivata automaticamente dal modulo del kernel che gestisce il bus ATA, insieme alla modalità di trasferimento dati controllabile con l'opzione `-X`. Può capitare comunque che per alcuni dischi ed alcuni chipset questa funzionalità

possa causare problemi o non sia impostata correttamente al valore migliore, nel qual caso si può disabilitare l'accesso in DMA con questa opzione, ed impostare esplicitamente una modalità di trasferimento dati con `-X`.

Parametro	**Opzione**	**Significato**
look-ahead	`-A`	legge e imposta la modalità di *read-ahead* hardware, normalmente abilitata in modo automatico, (1 abilitata, 0 disabilitata).
readahead	`-a`	legge e imposta il numero di settori in *read-ahead*.
busstate	`-b`	legge e imposta lo stato del bus; il valore 1 indica un bus attivo, 0 un bus disattivo.
APM_level	`-B`	legge e imposta le modalità di risparmio energetico con un valore da 1 a 255; un valore più basso implica maggiore risparmio energetico, fino a 127 il disco può essere anche spento, con 255 si disabilita il risparmio energetico.
IO_support	`-c`	legge e imposta la modalità di I/O a 32 bit; il valore 0 la disattiva, il valore 1 la attiva in modalità standard, il valore 3 la attiva con una funzione di sincronizzazione richiesta da alcuni chipset.
using_dma	`-d`	legge e imposta l'uso del DMA per le operazioni di I/O su disco; il valore 0 la disattiva, il valore 1 la attiva.
keepsettings	`-k`	legge ed imposta il flag di mantenimento delle impostazioni del disco effettuate con le opzioni `-d`, `-m`, `-u`, un valore 0 le scarta in caso di reset o spegnimento, un valore 1 le mantiene.
multcount	`-m`	legge ed imposta il numero di settori trasferiti nella singola operazione di I/O; il valore 0 disabilita il trasferimento multiplo, prende valori potenze di due fino ad un massimo che dipende dal supporto del disco.
acoustic	`-M`	legge ed imposta il livello di rumorosità, secondo le funzionalità AAM (*Automatic Acoustic Management*) presenti nei dischi recenti, prende un valore intero da 0 a 254, con un valore più basso che richiede una minore rumorosità, ottenuta diminuendo la velocità di rotazione dei dischi a scapito delle prestazioni.[13]
max sectors	`-N`	legge ed imposta il numero massimo di settori visibili, una funzionalità, chiamata *Host Protected Area*, che consente di mascherare una parte del disco al sistema operativo, di solito usata dai costruttori per mantenervi software diagnostico o di ripristino; i cambiamenti, a meno di non anteporre la lettera "`p`" al valore specificato, sono temporanei e vengono persi con un riavvio o un reset del disco.
readonly	`-r`	legge ed imposta il flag di sola lettura per il disco, il valore 0 disattiva la sola lettura, il valore 1 la attiva.
unmaskirq	`-u`	legge ed imposta il flag di blocco degli interrupt durante le operazioni di I/O; il valore 0 mantiene il blocco, il valore 1 lo disattiva.
write-caching	`-W`	legge ed imposta l'uso della cache di scrittura dei dischi; il valore 0 la disattiva, il valore 1 la attiva.

Tabella 6.5: Principali parametri accessibili con `hdparm` in lettura e scrittura e relative opzioni.

Un'altra opzione importante per le prestazioni è `-m` che imposta il numero di settori ottenibili in un singolo trasferimento di dati in DMA, una funzionalità presente in molti dischi moderni, che consente di ottenere una maggiore velocità di trasferimento dati, con un numero inferiore di interrupt di I/O. A seconda dei casi si possono ottenere miglioramenti fra il 5 ed il 50%, anche se con alcuni dischi si sono riscontrati peggioramenti. I valori che si possono specificare per questa

[13] la gran parte dei dischi supporta solo un numero di stati ridotto, spesso soltanto due `fast` (massime prestazioni) e `quiet` (bassa rumorosità) oltre eventualmente uno stato di spegnimento; per questi sono stati riservati rispettivamente i valori 254, 128 e 0; ma essendo prevista la possibilità di estensioni l'opzione permette l'uso di un valore intero generico.

opzione dipendono dal tipo di disco, ed in genere variano in potenza di 2 da 2 ad un massimo che può essere ottenuto dalle caratteristiche del disco usando l'opzione `-i` (viene riportato nell'output del comando alla voce *MaxMultSect*). In genere valori di 16 o 32, se supportati, sono ottimali.

Un'altra opzione significativa rispetto alle prestazioni è `-u`, con cui si può eliminare il mascheramento degli interrupt durante l'I/O su disco; il default (con il valore 0) prevede infatti il blocco di altri interrupt, cosa che può causare delle latenze nella risposta del sistema; eliminandolo (col valore 1) si migliora la reattività del sistema al costo di una minore velocità di trasferimento. Questa opzione è però potenzialmente pericolosa dato che alcuni chipset possono non tollerare l'aumento delle latenze nell'I/O su disco, dando luogo a perdita di dati con conseguente corruzione del filesystem.

Si tenga presente però che i cambiamenti effettuati con `hdparm` non sono permanenti, ed eventuali impostazioni vengono perse oltre che con lo spegnimento anche in caso di reset del disco; questo come salvaguardia in caso di impostazioni sbagliate; è possibile però richiedere ai dischi, usando l'opzione `-k`, di preservare le impostazioni effettuate con le opzioni `-d`, `-m`, `-u` così da non doverle rieffettuare; ovviamente si deve evitare di attivare questa funzionalità fintanto che non si è certi della validità delle impostazioni correnti. Inoltre è possibile su alcune distribuzioni utilizzare il file di configurazione `hdparm.conf` per mantenere le impostazioni che dovranno essere eseguite al riavvio della macchina, per il formato si consulti la relativa pagina di manuale.

Un elenco degli altri principali parametri modificabili con specifiche opzioni di `hdparm` nelle modalità appena descritte, con una breve spiegazione del relativo significato è illustrato in tab. 6.5; in prima colonna si è riportato il nome utilizzato dal comando nella stampa dei valori del parametro ed in seconda colonna la opzione corrispondente.

Opzione	Significato
`-g`	mostra la geometria del disco, se applicato ad una partizione il numero di settori ed il settore di inizio.
`-i`	stampa le informazioni sul dispositivo come viste dal kernel.
`-I`	stampa le informazioni sul dispositivo interrogando direttamente lo stesso sul bus.
`-t`	esegue una stima delle prestazioni di lettura del disco.
`-T`	esegue una stima delle prestazioni di trasferimento dati con il bus.
`-X`	imposta la modalità di trasferimento dati per i dischi, utilizzando come argomento uno dei valori ottenuti con l'uso dell'opzione `-i`.[14]
`-z`	forza il kernel ad eseguire una rilettura della tabella delle partizioni.

Tabella 6.6: Principali opzioni generiche del comando `hdparm`.

Oltre alle modifiche dei parametri appena illustrate, `hdparm` supporta una serie di ulteriori comandi diagnostici e di controllo. In particolare con l'opzione `-i` si possono ottenere una serie di informazioni relative al dispositivo, comprensive di nome del modello, numero di serie, geometria, modalità del trasferimento dati, ecc. così come viste dal kernel; se invece si usa l'opzione `-I` le informazioni vengono richieste direttamente al dispositivo stesso tramite il bus, e stampate.[15]

[14] in particolare nelle righe *PIO mode*, *DMA mode* e *UDMA mode*, si tenga presente comunque che tutti i dischi moderni riconoscono ed utilizzano automaticamente la migliore modalità di trasferimento disponibile, e che modificare queste impostazioni può essere molto pericoloso portare ad una massiccia corruzione dei dati.

[15] questa opzione, nella forma `--Istdin`, consente di stampare le stesse informazioni leggendole dallo *standard input*, nel formato prodotto dal comando stesso con l'opzione `--Istdout`, o da `sdparm` con l'opzione `-HHH`.

Altre opzioni interessanti sono `-t` e `-T` che permettono di eseguire delle stime sulle prestazioni dei dischi; si tenga presente che per una minima attendibilità occorre eseguire più volte questi test su un sistema inattivo. Con `-t` si ha una stima della massima velocità di lettura dal disco, mentre con `-T` si stima la massima velocità di trasferimento fra processore, memoria e bus della macchina.

Si sono riassunte le più comuni fra queste altre opzioni generiche in tab. 6.6, oltre a queste `hdparm` prevede molte altre opzioni che consentono di intervenire a basso livello sulle diverse funzionalità dell'interfaccia IDE/ATA; in quasi tutti i casi però l'impostazione di valori sbagliati può condurre a conseguenze disastrose con il rischio di una corruzione massiccia del filesystem ospitato sul disco, pertanto si lascia la trattazione di tutte le opzioni specialistiche alla pagina di manuale.

Benché alcune funzionalità siano accessibili anche attraverso `hdparm`, in maniera abbastanza completa per i dischi SATA, ma molto limitata nel caso di dischi SCSI o chiavette USB che non supportano le modalità di compatibilità, il protocollo SCSI prevede dei meccanismi generici per inviare comandi di controllo ai dispositivi, che questi siano dischi, CD, nastri. Per questo motivo è disponibile un apposito comando `sdparm` che, se usato con i dischi, consente l'accesso ai parametri e l'invio di comandi di basso livello in maniera analoga ad `hdparm`, usando direttamente il protocollo SCSI.

La comprensione del funzionamento di `sdparm` richiede però un minimo di conoscenza dei meccanismi previsti dal protocollo SCSI per mantenere le informazioni e modificarle, il che rende l'uso del comando piuttosto complesso. Le informazioni relative ad un dispositivo sono infatti disponibili in due modalità diverse; il primo è quello delle cosiddette pagine VPD (*Vital Product Data*), che contengono soltanto informazioni descrittive non modificabili. Il secondo meccanismo è quello delle cosiddette *mode pages* che contengono informazioni e parametri che si possono modificare per effettuare le impostazioni volute.

In entrambi i casi la quantità di informazione è estremamente ampia e non è possibile documentare in questa sede i dettagli disponibili per l'infinita serie di parametri accessibili al comando (che sono oltre 500, distribuiti fra le pagine VPD e *mode pages*). Per questo motivo lo stesso `sdparm` supporta una opzione generica `-e` (o `--enumerate`) che consente di elencare tutte le pagine presenti nelle sue tabelle interne, associandole a degli opportuni identificativi alfanumerici, che poi potranno essere usati per farvi riferimento; in questo caso il comando ignora un eventuale file di dispositivo passato come argomento.

Se invocato solo con `-e` il comando stampa soltanto la lista delle cosiddette *mode pages* "*generiche*", cioè quelle attinenti funzionalità generali indipendenti dal tipo di hardware utilizzato. Oltre a queste infatti esistono sia delle *mode pages* specifiche attinenti il singolo dispositivo SCSI, create dal produttore, che delle *mode pages* associate alle varie tecnologie di interconnessione dei dispositivi, come lo SCSI classico, il *Fiber Channel*, il *Serial Attached SCSI*, il *Serial ATA*. Un esempio del risultato di questo comando è il seguente:

```
anarres:~# sdparm --enumerate
Mode pages:
  bc   0x1c,0x01  Background control (SBC)
  ca   0x08       Caching (SBC)
  cms  0x2a       CD/DVD (MM) capabilities and mechanical status (MMC)
  co   0x0a       Control
  coe  0x0a,0x01  Control extension
  ...
```

in cui, sotto una riga di intestazione che descrive a quale gruppo di pagine si fa riferimento, viene stampato l'elenco delle pagine presenti su tre colonne che riportano rispettivamente un identificativo alfanumerico (in genere l'acronimo del nome esteso), il codice numerico ed un nome esteso della pagina, usualmente quello con cui questa viene descritta nei standard di riferimento.

Per ottenere invece la lista delle pagine VPD occorre aggiungere a `-e` l'opzione `-i` (o `--inquiry`). Se invece a `-e` si aggiunge l'opzione `-l` (o `--long`) si otterrà la lista completa di tutte le *mode pages* presenti, ed anche la lista dei produttori e dei meccanismi di trasporto riconosciuti dal comando a loro volta associati a degli identificativi alfanumerici. Si possono poi usare in combinazione con `-e` le opzioni `-M` (o `--vendor`) e `-t` (o `--transport`) per restringere la selezione rispettivamente alle *mode pages* relative ad un produttore o a una tecnologia di trasporto, indicati con i relativi identificativi alfanumerici (quelli che si ottengono con `sdparm -e -l`). Infine l'uso dell'opzione `-a` (o `--all`) fa stampare invece degli identificativi delle pagine quelli associati ai singoli parametri in essa contenuti.

Una volta che si siano identificate le pagine che interessano, si potranno ottenere le informazioni ed i valori dei parametri in esse contenuti utilizzando l'opzione `-p` (o `--page`); questa prende come argomento o l'identificativo alfanumerico o il codice numerico ottenuti con `-e` e stampa tutti i parametri ed i relativi valori presenti nella pagina stessa. Si è riportato in tab. 6.7 un elenco delle pagine più rilevanti; si tenga presente infine che quando si richiedono i dati di una pagina è necessario passare al comando il dispositivo di cui si vogliono ottenere i parametri come argomento.

Pagina	**Significato**
`di`	*Device identification*, la pagina VPD che contiene le informazioni con i dati identificativi del dispositivo, stampata di default con l'opzione `-i`.
`sv`	*Supported VPD pages*, la pagina che contiene la lista delle pagine VPD supportate.
`rw`	*Read write error recovery*, la *mode page* che contiene i parametri relativi alla gestione degli errori di accesso.
`ca`	*Caching*, la *mode page* che contiene i parametri relativi alla gestione della cache dei dati.
`co`	*Control*, la *mode page* che contiene i parametri di controllo del dispositivo.
`po`	*Power condition*, la *mode page* che contiene i parametri di controllo del risparmio energetico.

Tabella 6.7: Le *mode pages* e le pagine VPD più rilevanti.

Data la complessità con cui le informazioni sono distribuite nelle varie pagine `sdparm`, se usato senza nessuna opzione e passando soltanto l'argomento del file di dispositivo, stampa una selezione dei parametri più significativi relativi alle *mode pages* più rilevanti; ad esempio per un disco *Serial ATA* si otterrà qualcosa del tipo:

```
anarres:~# sdparm /dev/sda
    /dev/sda: ATA       ST3250310AS       3.AA
Read write error recovery mode page:
  AWRE        1
  ARRE        0
  PER         0
Caching (SBC) mode page:
```

```
    WCE        1
    RCD        0
  Control mode page:
    SWP        0
```

in cui nella prima riga vengono riportati i dati di base del dispositivo, ed a seguire i nomi estesi delle *mode pages* ed al di sotto su due colonne gli identificativi alfanumerici ed i valori dei parametri.

In genere nella lettura si fa uso anche dell'opzione `-l` che consente di ottenere una terza colonna con i nomi estesi dei parametri (di cui generalmente gli identificativi costituiscono un acronimo), se poi si vogliono ottenere tutte le *mode pages* e tutti i parametri disponibili per il dispositivo, si può usare l'opzione `-a` (o `--all`). Per accedere alle informazioni mantenute nelle pagine VPD si deve invece usare l'opzione `-i` che da sola stampa dati di identificazione del dispositivo (cioè il contenuto della pagina `di`). Se la si usa unita a `-a` stampa invece l'elenco delle pagine VPD disponibili, vale a dire il contenuto della pagina `sv`.

In genere si usano le informazioni delle pagine VPD per ottenere i dati relativi ai vari dispositivi: in sostanza l'opzione `-i` di `sdparm` è l'equivalente delle opzioni `-i` e `-I` di `hdparm`. Non esistono invece delle opzioni specifiche relative alle impostazioni dei dispositivi in quanto, come accennato, queste vengono effettuate modificando gli opportuni parametri nelle *mode pages*. A questi, oltre ad un valore corrente, è associato un flag che indica la possibilità o meno di modificarlo,[16] un valore di default stabilito dal produttore ed un valore *salvato*, che è quello che verrà usato al successivo reset del dispositivo stesso. Le operazioni di impostazione avvengono indicando a `sdparm` quale parametro modificare usando il relativo identificativo alfanumerico (quello che si può ottenere con le opzioni viste finora).

Le opzioni che consentono di operare sul singolo parametro sono tre: con `-g` (o `--get`), seguita dall'identificativo del parametro, se ne ottiene il valore, con `-s` (o `--set`) se ne imposta il valore, da specificare in forma di assegnazione all'identificativo del parametro: i valori sono sempre numerici, ed espressi in forma decimale, a meno di non apporvi il prefisso `0x` per usare la notazione esadecimale. Infine se a questa si abbina `-S` (o `--save`) il valore specificato verrà impostato oltre che come valore corrente anche come valore salvato. Si possono poi ripristinare tutti i parametri di una *mode page* (da indicare con `-p`) ai rispettivi valori di default usando l'opzione `-D` (o `--defaults`).

Parametro	**Significato**
`AWRE`	*Automatic Write Reallocation Enabled* controlla la riallocazione automatica dei settori danneggiati in caso di errori di scrittura (1 abilita, 0 disabilita, con default 1).
`ARRE`	*Automatic Read Reallocation Enabled* controlla la riallocazione automatica dei settori danneggiati in caso di errori di lettura (1 abilita, 0 disabilita, con default 1).
`WCE`	*Write Cache Enable* controlla la cache di scrittura, (1 abilita, 0 disabilita, con default 1), è equivalente all'opzione `-W` di `hdparm`.
`DRA`	*Disable Read Ahead*, controlla il *read-ahead* del disco, (1 disabilita, 0 abilita, con default 0), è equivalente all'opzione `-A` di `hdparm`.

Tabella 6.8: I parametri più rilevanti per le impostazioni con `sdparm`.

[16]alcuni parametri sono determinati dallo stato del dispositivo e non sono modificabili dall'utente.

Così se si vuole impostare il *write caching* del disco (equivalente dell'opzione `-W` di `hdparm`) si potrà usare il comando

```
sdparm --set WCE=1 /dev/sda
```

e per effettuare un test delle impostazioni (in cui si esegue un controllo della validità dei valori richiesti) senza eseguirle effettivamente si può aggiungere l'opzione `-d` (o `--dummy`). Come per le *mode pages* e le pagine VPD anche solo fare elenco di tutti i possibili parametri di configurazione e del loro significato va al di là delle possibilità di questo testo; si sono riportati in tab. 6.8 alcuni fra i più rilevanti.

Infine `sdparm` permette di inviare ai dispositivi, attraverso il protocollo SCSI, una serie di comandi specifici usando l'opzione `-C` (o `--command`); a questa deve seguire come parametro il nome del comando. I comandi definiti, che non è detto siano applicabili a qualunque dispositivo (alcuni infatti sono specifici dei masterizzatori) sono riportati in tab. 6.9 insieme alla spiegazione del relativo significato.

Comando	**Significato**
`capacity`	richiede la capacità del supporto, riportando il numero di blocchi, la loro dimensione e la dimensione totale in Mb.
`eject`	ferma il supporto (se in rotazione) e lo espelle.
`load`	carica il supporto e lo mette in rotazione.
`ready`	esegue una interrogazione per verificare se il dispositivo è pronto ad una risposta (cosa che non avviene se ad esempio si è fermato un disco).
`sense`	richiede alcune informazioni relative al dispositivo,[17] come le informazioni sull'alimentazione o lo stato di progresso di un comando lento (come una formattazione).
`start`	avvia il supporto, mettendolo in rotazione.
`stop`	ferma il supporto, bloccandone la rotazione, è equivalente a `hdparm -y`.
`sync`	scarica i buffer interni del dispositivo con l'immediata scrittura dei dati sul supporto, è equivalente a `hdparm -f`.
`unlock`	indica al dispositivo di consentire la rimozione del supporto (da usare come ultima risorsa su un dispositivo bloccato), equivalente a `hdparm -L0`.

Tabella 6.9: I nomi dei comandi per l'opzione `-C` di `sdparm`.

6.3.2 Il ridimensionamento di filesystem e partizioni

Abbiamo già incontrato in sez. 6.1.3, con l'accrescimento di un RAID software, ed in sez. 6.2.4, con la possibilità di modificare le dimensioni di un volume fisico di LVM, due casi in cui un dispositivo contenente un filesystem può cambiare dimensioni. In entrambi i casi si è sottolineato come questo cambiamento non comporti anche l'adattamento del filesystem alle nuove dimensioni; questo infatti quando viene creato organizza le sue strutture interne assumendo che il dispositivo su cui si trova abbia una certa dimensione, che è quella rilevata a quel momento.

Per questo motivo le operazioni di modifica delle dimensioni di un dispositivo, che sia questo un RAID software, un volume logico di LVM, o semplicemente una partizione di cui si sia spostata la fine con i comandi di sez. 5.1.2, devono essere sempre accompagnate dalle conseguenti

[17] si tratta in questo caso di uno specifico comando del protocollo, denominato *REQUEST SENSE SCSI*, che richiede alcuni dati relativi al dispositivo detti appunto *sense data*.

operazioni di modifica del filesystem sottostante. Ed in particolare si deve tener conto che se aumentare le dimensioni di un dispositivo non ha nessun effetto sul filesystem in esso contenuto, il ridurle senza prima aver ridotto anche quelle del filesystem comporta che questo non sia più in grado di accedere a parte delle sue strutture (quelle poste nella parte tolta) con conseguente immediata corruzione dello stesso e probabile perdita di dati.

Pertanto quando si vanno ad effettuare operazioni di ridimensionamento è assolutamente necessario ricordarsi che la corretta sequenza di operazioni non è la stessa se si compie un allargamento o una riduzione. Nel primo caso infatti occorre prima allargare il dispositivo e poi il filesystem, mentre nel secondo occorre prima ridurre il filesystem e poi il dispositivo.

La possibilità di ridimensionare un filesystem dipende dalla presenza di un opportuno programma, specifico per ciascun tipo di filesystem, in grado di svolgere questo compito. Correntemente la gran parte dei filesystem presenti su Linux (ext2/ext3/ext4, reiserfs, XFS, JFS) è dotata di questa funzionalità. In generale questa operazione può essere eseguita solo a filesystem inattivo, cioè senza che questo sia montato, il che comporta degli ovvi problemi se si vuole ridimensionare quello contenente la directory radice. Alcuni filesystem però supportano anche il ridimensionamento *a caldo* (cioè senza smontare il filesystem) se questo avviene in accrescimento.

Come accennato ogni filesystem ha normalmente un suo programma per il ridimensionamento, fa eccezione JFS, che utilizza una specifica opzione di montaggio; per ridimensionare un filesystem JFS infatti basterà rimontarlo con un comando del tipo:

```
mount -t jfs -o remount,resize=10G /dev/vg/home /home
```

dove se non si specifica nessun parametro per l'opzione `resize` il ridimensionamento viene fatto alla dimensione totale del dispositivo sottostante. Il ridimensionamento può essere eseguito a caldo come nell'esempio, ma si tenga presente che JFS, come XFS, non supporta il restringimento del filesystem, le dimensioni possono soltanto essere aumentate.

Nel caso di XFS il comando che consente il ridimensionamento del filesystem è `xfs_growfs`, il comando richiede che si passi come argomento il *mount-point* del filesystem, e funziona soltanto a caldo, vale a dire richiede che il filesystem sia montato. Come per JFS non è possibile restringere un filesystem XFS. Se non si specifica nulla il ridimensionamento avviene al totale della nuova dimensione del dispositivo, per le opzioni si consulti al solito la pagina di manuale.

Per ridimensionare un filesystem *reiserfs* si deve invece usare il comando `resize_reiserfs`, passando come argomento il dispositivo contenente il filesystem da ridimensionare. La nuova dimensione si specifica con l'opzione `-s` seguita dal valore, con la solita convenzione dell'iniziale per le unità di misura per indicare kilobyte, megabyte e gigabyte. L'opzione supporta anche l'uso dei segni "`+`" e "`-`" per indicare dei ridimensionamenti relativi (accrescimenti o riduzioni); se non si specifica nulla il comando userà la dimensione del dispositivo sottostante.

Il comando permette anche di specificare un dispositivo alternativo su cui mantenere il giornale con l'opzione `-j`, e di forzare le operazioni senza eseguire controlli con `-f`; per avere più informazioni nel corso delle operazioni si deve invece usare `-v`. In caso di accrescimento il il comando può essere usato a caldo, mentre in caso di riduzione è necessario smontare il filesystem.

Infine il comando che permette di ridimensionare un filesystem *ext2*, *ext3* o *ext4*, è `resize2fs`. A seconda delle versioni del filesystem il programma può richiedere che il filesystem sia smontato anche in caso di accrescimento, per poter fare l'accrescimento a caldo infatti il filesystem deve essere montato almeno come *ext3* e deve essere stato creato con la presenza della estensione `resize_inode`, vedi tab. 5.6.

Il comando prende come primo argomento il dispositivo su cui si trova il filesystem da ridimensionare. Se non viene specificato un secondo argomento indicante la dimensione da usare il ridimensionamento avviene alla dimensione corrente del dispositivo, cosa che significa che l'argomento non è necessario in caso di espansione e lo è in caso di riduzione. La dimensione può essere indicata con la convenzione sulle unità di misura già illustrata in precedenza che prevede che queste siano specificate posponendo al numero la relativa iniziale.

Il comando prevede l'opzione `-d` per abilitare il debug, che prende come parametro una bitmask indicante quali informazioni stampare, con l'opzione `-p` stampa una percentuale di completamento durante le operazioni, mentre `-f` forza le operazioni escludendo alcuni controlli di sicurezza e `-F` scarica i dati presenti in cache prima di iniziare.

Si tenga presente che anche se i comandi citati sono in genere in grado di determinare da soli la dimensione di un dispositivo, qualora se ne specifichi manualmente una eccessiva, si andrà incontro a problemi. Inoltre una volta che sia ridotto un filesystem e si proceda al restringimento del dispositivo sottostante, occorre stare molto attenti a non farlo al di sotto delle dimensioni date al filesystem.

Infine quando si ridimensiona una partizione per poi ridimensionare il filesystem occorre anche avere cura che essa continui ad avere lo stesso punto di inizio, altrimenti i comandi non potranno più trovare i dati del filesystem. Se questo avviene automaticamente per il RAID e LVM, non altrettanto accade quando si usa `fdisk` come abbiamo visto in sez. 5.1.2 per la manipolazione del contenuto della tabella delle partizioni.

Quando su un disco sono già presenti dei dati questo in genere significa che al più si potranno cancellare alcune delle partizioni per poi riutilizzare lo spazio ottenuto o al più allargare o stringere una partizione esistente, posto che nel primo caso si sia opportunamente ristretto il filesystem in esso contenuto, e che nel secondo sia disponibile dello spazio libero in coda alla partizione e si allarghi poi il filesystem.

Per questo motivo quando si opera direttamente sulle partizioni è più opportuno usare un comando come `parted` che unisce le funzionalità dei comandi per il partizionamento e di quelli per il ridimensionamento dei file (ed anche di quelli di controllo e riparazione) e consente non solo la creazione e la cancellazione delle partizioni, ma anche il loro ridimensionamento e spostamento senza perderne il contenuto. Una caratteristica interessante del programma è che ne è disponibile una versione su disco di avvio che consente di ripartizionare un sistema prima di avviarlo, cosa che permette di operare anche sulla partizione che contiene il filesystem della radice.

Il programma opera normalmente in modalità interattiva, e lo si può lanciare direttamente ottenendo un prompt da cui invocare i comandi interni; se non si specifica nessun argomento esso andrà ad operare sul primo disco, altrimenti il primo argomento deve essere il file di dispositivo su cui si vuole operare, a questo può seguire, per una invocazione non interattiva, la riga di comando che si darebbe dal prompt, ma in questo caso il file di dispositivo deve essere sempre indicato.

La gran parte dei comandi interni richiedono il passaggio di opportuni argomenti, il più comune è il numero della partizione su cui operare, che è quello che abbiamo illustrato in sez. 5.1.2. Quando è richiesta l'indicazione di un disco questa deve essere effettuata specificando il relativo file di dispositivo, mentre l'indicazione di un tipo di filesystem utilizza gli stessi valori usati per l'opzione `-t` di `mount`, ma si tenga presente che le operazioni sui filesystem sono supportate solo per alcuni di essi.

Infine qualora sia richiesta la specificazione di una dimensione, che questa sia un punto di inizio, di fine o l'estensione di una partizione, il default del programma prevede che un valore numerico sia interpretato come numero di megabyte. Si possono però usare anche altre unità di misura apponendo al numero i relativi suffissi (ad esempio `kB` per kilobyte e `GB` per gigabyte), si possono inoltre specificare le dimensioni in percentuale con il suffisso `%` ed usare numeri negativi per indicare una dimensione a partire dal fondo del disco. Tutte le indicazioni relative alle dimensioni vengono considerate come approssimate, e vengono automaticamente aggiustate in un intervallo di 500Mb per rispettare la geometria del disco, a meno di non specificare un valore esatto in numero di settori, usando il suffisso `s`.

Il comando principale di `parted` è `resize`, che consente di ridimensionare e spostare una partizione; il comando prevede che si specifichi la partizione su cui operare seguita dal punto di inizio e da quello di fine. Il comando non modifica mai il numero della partizione, ma posto che ci sia lo spazio disponibile sposta i dati e ridimensiona il filesystem per far si che la partizione indicata inizi e termini nei punti specificati. Si tenga presente che la combinazione di uno spostamento ed di un ridimensionamento non è supportata per tutti i filesystem, nel qual caso sarà necessario, prima eseguire il ridimensionamento senza cambiare il punto di inizio e poi eseguire lo spostamento della partizione con `move`.

Comando	**Significato**
`rm`	cancella una partizione, se si cancella una partizione logica tutte quelle di numero superiore verranno rinominate per coprire il "*buco*".
`print`	stampa la tabella delle partizioni ed alcuni dati riassuntivi sul disco.
`cp`	copia una partizione, prende come primo argomento opzionale il dispositivo sorgente (da specificare con il rispettivo file di dispositivo), i due argomenti successivi indicano la partizione sorgente e quella di destinazione (che fa sempre riferimento al disco corrente).
`rescue`	recupera una partizione cancellata accidentalmente, richiede che si specifichi approssimativamente il punto di inizio e di fine della stessa, e se identificata chiederà il numero da assegnargli.
`resize`	ridimensiona una partizione, richiede il numero della partizione, il punto di inizio e di fine; in caso di estensione lo spazio ulteriore deve essere libero.
`select`	consente di selezionare un altro disco; usato in modalità interattiva.
`move`	sposta una partizione, richiede il numero della stessa seguito dal nuovo punto di inizio e di fine; lo spazio dove la partizione viene spostata deve essere libero.
`unit`	cambia l'unità di misura delle dimensioni, usata in modalità interattiva.

Tabella 6.10: I principali comandi di `parted`.

Un altro comando interessante di `parted` è `rescue` che consente di recuperare una partizione cancellata per errore, il comando richiede una indicazione approssimata dei punti di inizio e fine della stessa, ed esegue una ricerca. Una volta identificati i limiti esatti chiede il numero da assegnare alla partizione recuperata.

Si sono riportati in tab. 6.10 i comandi di `parted` più significativi, insieme ad una breve descrizione del significato e degli argomenti necessari; si tenga presente che quando un comando è invocato in modalità interattiva eventuali argomenti mancanti verranno chiesti sul terminale. Oltre a quelli citati il `parted` supporta anche altri comandi per eseguire il partizionamento o la creazione dei filesystem che duplicano le funzionalità dei comandi standard già visti in sez. 5.1 e che perciò non tratteremo. L'elenco può comunque essere ottenuto dalla pagina di

manuale, anche se documentazione completa del comando è fornita soltanto nelle pagine *info* (vedi sez. 2.4.2), accessibile con `info parted`.

6.3.3 Dischi e filesystem cifrati

Una delle problematiche ricorrenti relative alla sicurezza dei dati è la possibilità di avere una protezione degli stessi anche in caso di smarrimento o furto della macchina o dei supporti su cui sono memorizzati. L'accesso fisico ad un disco infatti consente a chiunque di leggerne il contenuto, per cui non si può assumere che questo accesso sia effettuato sempre sotto il controllo del proprio sistema operativo.

In questo caso l'unica misura di protezione efficace è la cifratura dei dati, in modo che essi possano essere riletti soltanto dopo aver effettuato una opportuna procedura di decifrazione, ovviamente subordinata ad una identificazione (come l'immissione di una password di sblocco) del richiedente, in assenza della quale anche il possesso fisico del supporto rende impossibile l'accesso ai dati stessi. Ovviamente questo non garantisce nulla fintanto che questi saranno accessibili (ad esempio in caso di compromissione della macchina su cui si trovano) ma soltanto dalla impossibilità di poterli riutilizzare una volta che questi saranno stati resi indisponibili (spegnendo la macchina o disabilitando la decifrazione).

Le modalità in cui si può realizzare tutto ciò sono varie, fra queste spicca una delle applicazioni più interessanti dell'infrastruttura del *device mapper* del kernel, che abbiamo già incontrato a proposito di LVM, che consente di cifrare interamente il contenuto di un disco o di una partizione. In questo caso, invece di rimappare lo spazio disponibile sui dischi fisici in volumi logici, un dispositivo fisico viene rimappato su un volume cifrato. Questo significa che quanto verrà scritto in chiaro sul volume cifrato, verrà cifrato e salvato sul disco fisico, e viceversa si potranno rileggere i dati cifrati dal disco fisico ottenendoli in chiaro sul volume cifrato.

In questo caso la cifratura dei dati viene effettuata direttamente dal kernel ed avviene a livello di dispositivo. Questo consente il massimo delle prestazioni, ed alcune distribuzioni, come Debian e RedHat, offrono questa possibilità anche in fase di installazione. In tal caso tutte le impostazioni vengono fatte dal programma di installazione, e le password per abilitare l'accesso ai dischi cifrati verranno chieste automaticamente all'avvio del sistema operativo.

Dato che la creazione di un volume cifrato comporta solo la predisposizione della rimappatura dello stesso su uno specifico dispositivo fisico, si deve avere ben presente che i precedenti contenuti di quest'ultimo non vengono modificati dall'operazione. Per questo è necessario eseguire preventivamente una sovrascrittura completa dei dati del dispositivo fisico con valori casuali, con comandi come `shred`, `dd` o `badblocks`,[18] per evitare che si possano distinguere chiaramente dal contenuto precedente quali sono le nuove parti cifrate, che verranno sovrascritte solo coi successivi accessi tramite il nuovo volume cifrato.

Le tecnologie disponibili per la creazione di volumi cifrati con il *device mapper* sono sostanzialmente due, la cifratura immediata usando *dm-crypt* e l'uso di LUKS (*Linux Unified Key Setup*), ma entrambe vengono gestite attraverso lo stesso programma, `cryptsetup`. Esiste in realtà la possibilità di gestire con `cryptsetup` anche altre tecnologie di cifratura non connesse

[18] ad esempio, rifacendosi al dispositivo fisico `/dev/sdd1` si possono usare i comandi `badblocks -c 10240 -s -w -t random -v /dev/sdd1`, `dd if=/dev/urandom of=/dev/sdd1` o `shred /dev/sdd1` in ordine crescente di paranoia (e di tempo impiegato per eseguire l'operazione).

con il *device mapper*, come *loop-AES* e *TrueCrypt*, che non tratteremo. La sintassi generale del programma prevede che esso sia specificato nella forma:

```
cryptsetup comando [opzioni] [argomenti]
```

in cui "`comando`" specifica l'azione che dovrà essere eseguita da `cryptsetup`, un elenco dei principali comandi disponibili è stato riportato in tab. 6.11.

Comando	**Significato**
`open`	crea un volume cifrato del tipo indicato dall'opzione `--type`, richiede come primo argomento il file di dispositivo sottostante e come secondo argomento il nome del volume cifrato (che comparirà sotto `/dev/mapper`), richiede che si fornisca la chiave di accesso.
`create`	attiva un volume cifrato con *dm-crypt*, equivalente di `open` con `--type plain`.
`close`	rimuove un volume cifrato, eliminando la relativa mappatura, richiede come argomento il nome dello stesso, come compare sotto `/dev/mapper`.
`luksFormat`	crea l'intestazione di LUKS sul dispositivo passato come primo argomento, un secondo argomento opzionale indica il file da cui leggere la passphrase di accesso (un "`-`" implica la lettura dallo *standard input*), se non indicato la passphrase viene richiesta due volte sul terminale.
`luksOpen`	attiva un volume cifrato con LUKS, equivalente di `open` con `--type luks`.
`luksAddKey`	aggiunge una nuova chiave di accesso (una passphrase) nel primo slot disponibile, richiede come argomento il dispositivo su cui operare, e chiede la passphrase sul terminale.
`luksClose`	sinonimo di `close`.
`luksDump`	stampa una lista del contenuto degli slot, consentendo di individuare quelli utilizzati.
`luksKillSlot`	cancella completamente il contenuto di uno slot eliminandone la chiave contenuta, richiede sempre, per evitare di restare senza accesso, di fornire una delle chiavi restanti.
`luksHeaderBackup`	salva una copia binaria della intestazione di LUKS del dispositivo indicato come argomento.
`luksHeaderRestore`	ripristina la copia binaria (ottenuta da un precedente `luksHeaderBackup`) sul dispositivo indicato.

Tabella 6.11: I principali comandi di `cryptsetup`.

La scelta del tipo di cifratura deve essere specificata con l'opzione `--type` (che prende i valori riportati in tab. 6.12) quando si crea il dispositivo cifrato, cosa che viene fatta con il comando `open`, che richiede come argomenti successivi il file di dispositivo del disco sottostante ed il nome da assegnare al volume cifrato. Esistono tradizionalmente delle "abbreviazioni" (utilizzate per compatibilità con le vecchie versioni del comando) come `create` che sottintende l'uso di *dm-crypt* e `luksOpen` che sottintende l'uso di LUKS.

Valore	**Tipo**
`plain`	cifratura semplice con *dm-crypt*.
`luks`	cifratura con LUKS.
`loopaes`	cifratura con *loop-AES*.
`tcrypt`	cifratura con *TrueCrypt*.

Tabella 6.12: Tipi di cifratura supportati da `cryptsetup` (argomenti per l'opzione `--type`).

Allora se ad esempio si vuole creare un dispositivo cifrato con *dm-crypt* utilizzando come supporto la partizione `/dev/sdd1` (si può usare anche un intero disco, un volume logico ed in generale un qualunque dispositivo a blocchi) si dovrà usare comando del tipo:

```
root@monk:~# cryptsetup create -y /dev/sdd1 nome
Enter passphrase:
Verify passphrase:
```

ed inserita la passphrase si potrà accedere al volume cifrato che verrà reso disponibile attraverso il dispositivo `/dev/mapper/nome`. Si tenga presente che la passphrase non viene mai mostrata sul terminale, per cui, per evitare il rischio di errori di battitura, la prima volta che si crea un volume cifrato prima di salvarci sopra dei dati, è opportuno utilizzare l'opzione `-y` che richiede una seconda immissione per verifica.

A questo punto si potrà accedere a `/dev/mapper/nome` come ad un disco qualunque, e ad esempio crearvi un filesystem, montarlo e salvarvi sopra i propri dati. Una volta che non ne sia più necessario l'utilizzo si potrà disattivare la mappatura del volume cifrato attraverso l'uso del comando `close` (ovviamente se montato dovrà prima essere smontato) che prende come unico argomento il nome dello stesso. Per riaccedervi successivamente basterà ripetere l'operazione di creazione del volume cifrato con il comando `open`, anche usando un nome diverso e senza la necessità di usare l'opzione `-y`.

Si tenga presente che *dm-crypt* esegue sempre una cifratura immediata settore per settore utilizzando la passphrase fornita in fase di creazione/attivazione del dispositivo cifrato, che viene usata direttamente come chiave dal sottostante algoritmo di cifratura.[19] Detta passphrase non viene mai salvata, non vengono fatti controlli di consistenza e non esistono metadati. Se si immette una passphrase sbagliata essa verrà comunque accettata ed utilizzata, ottenendo un volume cifrato che contiene dei dati inintelligibili.

Questo significa anche che non esiste modo di cambiare la passphrase di cifratura che non sia quello di ricifrare da capo tutto il dispositivo, usando un `dd` per leggere dal volume originario, cifrato con la passphrase di partenza, e scrivere su quello nuovo, cifrato con la nuova passphrase. Tutto ciò richiede o la presenza di un dispositivo fisico di appoggio di dimensioni equivalenti per fare l'operazione in due passi, oppure, se si fa l'operazione utilizzando lo stesso dispositivo, cosa possibile dato che `dd` legge e scrive un blocco alla volta, per cui prima decifra e poi sovrascrive con la nuova cifratura, che si accetti il rischio, se il comando viene interrotto per un qualsiasi motivo, di trovarsi il disco cifrato un po' con la nuova e un po' con la vecchia passphrase, con una perdita di dati praticamente scontata.

L'uso di LUKS fornisce invece un minimo di infrastruttura in più, in tal caso infatti la passphrase richiesta in fase di creazione del volume cifrato (chiamata *chiave* nella terminologia di LUKS) serve a cifrare la chiave effettivamente usata dall'algoritmo crittografico, che viene generata in maniera totalmente casuale e non è nota all'utente. LUKS salva questa informazione in una intestazione posta all'inizio del dispositivo, che può contenere fino ad otto di queste "*chiavi*", si potrà pertanto, anche se la chiave crittografica effettivamente usata è sempre la stessa, utilizzare fino ad otto passphrase per sbloccarla ed utilizzare il volume cifrato.

Questa differenza rende immediatamente LUKS più sicuro: con *dm-crypt* la necessità di usare direttamente come chiave crittografica la passphrase, che è anche quella che occorre memorizzare,

[19] detto algoritmo si può specificare usando l'opzione `--cipher`, ma non vi è alcun motivo per cambiare il default che è `aes-cbc-essiv:sha256`, qualora lo si faccia si abbia però cura di annotarselo, perché anch'esso non viene memorizzato ed usarne uno diverso comporta l'impossibilità di decifrare il contenuto.

rende infatti impossibile o molto difficile usare un valore completamente casuale. Con LUKS la passphrase serve solo a decifrare la chiave effettiva, che è completamente casuale e quindi è praticamente impossibile indovinarla. Inoltre LUKS, avendo la possibilità di utilizzare più "*chiavi*", consente anche di far usare credenziali diverse a persone diverse, e cambiarle senza dover ricifrare il dispositivo sottostante.

A differenza di quanto visto per *dm-crypt* qualora si voglia usare LUKS occorre prima preparare adeguatamente il dispositivo. Questo si fa usando il comando `luksFormat`, che crea l'intestazione con i dati necessari ed imposta la prima passphrase di accesso; un possibile esempio è il comando seguente:

```
root@monk:~# cryptsetup luksFormat -y /dev/sdd1
WARNING!
========
This will overwrite data on /dev/sdd1 irrevocably.

Are you sure? (Type uppercase yes): YES
Enter passphrase:
Verify passphrase:
```

che crea sul disco sottostante (nel caso `/dev/sdd1`) l'infrastruttura necessaria al successivo uso dello stesso come volume cifrato con LUKS.

Alternativamente all'immissione sul terminale, `luksFormat` e tutti gli altri comandi che richiedono l'uso di una passphrase possono utilizzare l'opzione `--key-file` per effettuare la lettura della stessa dal file indicato come parametro. Questo permette ad esempio di non bloccare l'avvio se si deve montare il volume cifrato e nessuno può inserire la passphrase sul terminale, ma ovviamente chiunque otterrà l'accesso al file sarà in grado di utilizzare il disco cifrato. Si può utilizzare questo meccanismo per mettere la chiave su un dispositivo rimovibile rendendo inaccessibile il volume cifrato in assenza della stesso, o per rendere inaccessibili a terzi dischi contenenti backup riposti in collocazioni remote.

Una volta creata l'infrastruttura iniziale si potrà aggiungere una nuova passphrase con il comando `luksAddKey`, che andrà ad inserire la stessa nel primo slot libero disponibile (fino ad un massimo di otto) a meno di non usare l'opzione `-key-slot` per indicarne esplicitamente uno. Il comando richiede come argomento il dispositivo fisico e l'immissione di una passphrase valida sul terminale, chiedendo poi la nuova passphrase da aggiungere; un esempio è il seguente:

```
root@monk:~# cryptsetup luksAddKey /dev/sdd1
Enter any existing passphrase:
Enter new passphrase for key slot:
Verify passphrase:
```

ed anche in questo caso si potrà fornire la passphrase di accesso su un file con l'opzione `--key-file` mentre se si vuole fornire la nuova passphrase su un file questo andrà passato come secondo argomento.

Una delle caratteristiche di sicurezza più interessanti fornite da LUKS, che lo rende ancor più resistente ai tentativi di forzatura della passphrase, è l'uso di un algoritmo crittografico (denominato PBKDF2) che prevede che si possa richiedere un tempo minimo comunque necessario a decifrare la chiave. Questo rallenta enormemente la ripetizione dei tentativi di accesso, rendendo sostanzialmente inutilizzabili i programmi di forzatura delle password senza avere ha un impatto significativo nell'uso da parte di una persona. Questa durata si può impostare in fase di

creazione di una passphrase (quindi con `luksFormat` o `luksAddKey`) con l'opzione `-i` seguita dal numero di millisecondi; il valore di default è di un secondo, ma lo si può aumentare a piacere.

Per rimuovere una passphrase occorre invece il comando `luksKillSlot`, che oltre al dispositivo fisico su cui operare richiede come secondo argomento il numero di slot da cancellare. Di nuovo verrà richiesta una passphrase di accesso (o si può usare un file con l'opzione `--key-file`), ma il comando non consente di cancellare lo slot su cui questa è memorizzata. L'elenco degli slot occupati può essere ottenuto usando il comando `luksDump` che stampa le informazioni dell'intestazione di LUKS, con un risultato del tipo:

```
root@monk:~# cryptsetup luksDump /dev/sdd1
LUKS header information for /dev/sdd1
...
Key Slot 0: ENABLED
        Iterations:             161615
        Salt:                   91 19 9a dd eb 7f 1e 28 e8 e0 c7 1d 36 3a fa b2
                                  fe 40 b1 26 77 d9 87 f8 36 5b db 71 a8 41 a3 fc
        Key material offset:    8
        AF stripes:                4000
Key Slot 1: DISABLED
Key Slot 2: DISABLED
Key Slot 3: DISABLED
Key Slot 4: DISABLED
Key Slot 5: DISABLED
Key Slot 6: DISABLED
Key Slot 7: DISABLED
```

Una volta creata l'intestazione di LUKS si potrà "aprire" il volume cifrato con il comando `luksOpen`, che è l'analogo del precedente `create`, in tal caso verrà chiesta una passphrase di accesso, fornita la quale si otterrà il volume cifrato. Proseguendo il precedente esempio si potrà usare il comando:

```
root@monk:~# cryptsetup luksOpen /dev/sdd1 nome
Enter passphrase for /dev/sdd1:
```

che una volta fornita una delle passphrase disponibili renderà disponibile il volume cifrato come `/dev/mapper/nome`,[20] e si potrà operare su questo come su un qualunque dispositivo a blocchi (quindi vi si potrà creare un filesystem, montarlo, smontarlo ecc.).

Qualora si voglia disabilitare l'accesso ad un volume cifrato si potrà rimuoverlo con il comando `close` (o anche `luksClose`, che è un sinonimo). Ovviamente perché il comando abbia successo occorre che il volume non sia utilizzato (ad esempio se contiene un filesystem questo non dovrà essere montato).

6.4 Le quote disco

Una delle caratteristiche avanzate della gestione dei dischi presenti fin dal kernel 2.0 è quella delle quote disco, un meccanismo che consente di limitare la quantità delle risorse su disco che i singoli utenti (o gruppi) possono usare. Tratteremo in questa sezione il loro utilizzo e le relative modalità di gestione.

[20] a differenza di *dm-crypt*, che usa sempre la passphrase fornita, il comando fallisce se non si specifica una passphrase corretta.

6.4.1 Visione generale

Una delle risorse principali che gli amministratori di sistema devono gestire è quella dello spazio disco; uno dei problemi più comuni è infatti quello del riempimento del disco, e questo avviene sovente perché alcuni utenti non si curano di controllare le risorse che usano, e finiscono per riempire tutto lo spazio disponibile, a scapito pure degli altri utenti.

Per questo motivo una delle caratteristiche presenti da sempre in sistemi multiutente è quella di poter imporre delle restrizioni allo spazio disco utilizzato dai singoli utenti. Nel caso specifico di un sistema unix-like questo è sempre stato realizzato attraverso le cosiddette *quote disco*, un meccanismo, realizzato nel kernel, che pone delle restrizioni alle risorse che un singolo utente può utilizzare.

L'idea è che in questo modo esiste un limite di occupazione delle risorse oltre il quale ogni utente non può andare. La struttura del meccanismo di controllo opera sia sulla quantità di spazio disco (cioè sul numero di *blocchi*) che sul numero di file (cioè sul numero di *inode*) da mettere a disposizione; le due risorse sono completamente indipendenti per cui è possibile superare il limite sul numero di file occupati pur avendo ancora spazio disco disponibile.

Le quote sono sempre calcolate per ogni singolo filesystem (che deve supportare il meccanismo) pertanto è usuale attivarle sul filesystem su cui si sono piazzate le home degli utenti. Con gli ultimi kernel il supporto è presente per tutti i filesystem principali. Si tenga presente comunque che se gli utenti possono scrivere su più filesystem le quote opereranno in maniera indipendente su ciascuno di essi, e dovranno perciò essere impostate separatamente.

Un'ultima caratteristica generale delle quote disco è che il controllo sulla occupazione delle risorse può avvenire sia a livello di un singolo utente (cioè in base all'*user ID*), che di un gruppo (cioè in base al *group ID*), che di entrambi. In pratica, si avrà la possibilità di utilizzare il disco per aggiungere un certo file se questo è consentito dalla quota dell'utente cui apparterrà il file in questione o se è consentito dalla quota di gruppo (sempre per il gruppo di appartenenza del file).

6.4.2 Configurazione del sistema delle quote

Per poter utilizzare le quote disco occorre anzitutto abilitare il relativo supporto nel kernel nella sezione `File systems` della configurazione. Del sistema esistono due versioni, entrambe supportate per compatibilità, anche se da anni è in uso soltanto la più recente. Si tenga presente che se si vogliono usare le quote con NFS occorre che sia stato opportunamente avviato il relativo demone per la gestione (vedi sez. 8.4.1).

Il secondo passo per utilizzare il controllo delle quote è segnalarne la presenza in fase di montaggio del filesystem, perché il kernel sia successivamente in grado di mantenere i dati relativi all'uso del disco ed applicare le restrizioni. Questo viene fatto utilizzando due opzioni specifiche per `/etc/fstab`: per le quote degli utenti si usa `usrquota` mentre per quelle dei gruppi `grpquota`, un esempio di utilizzo di queste opzioni è il seguente:

/etc/fstab
```
/dev/vg/home    /home    ext3  defaults,usrquota,grpquota       0       1
```

Una volta abilitato il supporto nel kernel e montato opportunamente il filesystem il passo successivo è quello di avere presenti nella radice del relativo filesystem (la directory corrispon-

dente al *mount point*) i file `quota.user` e `quota.group` se si usa la versione 1 del supporto o i file `aquota.user` e `aquota.group` se si usa la versione 2. È in questa coppia di file che sono mantenuti i dati relativi all'uso del disco da parte di utenti e gruppi usati dal sistema delle quote. Questo non è vero per le quote su XFS, in detto filesystem infatti le quote sono mantenute nei meta-dati interni e non su file visibili.

In generale non è necessario creare questi file a mano anche se alcuni HOWTO indicano di crearli vuoti con il comando `touch` se non sono presenti. Se assente infatti la prima volta che si usa il comando `quotacheck` per controllare una quota il corrispondente file viene creato automaticamente. Il comando deve essere eseguito su un filesystem montato con le opzioni per le quote prima della loro abilitazione con `quotaon`, in modo da ricostruire lo stato dell'occupazione delle risorse sul disco. Se infatti le informazioni di `aquota.user` e `aquota.group` non sono aggiornate il meccanismo delle quote non funziona in maniera corretta.

Il comando prende come argomento il file di dispositivo su cui è posto il filesystem da controllare o il rispettivo *mount point* presente in `/etc/fstab`, ma questo argomento può essere tralasciato con l'uso dell'opzione `-a` che controlla tutti i filesystem montati, tranne quelli montati via NFS.

Opzione	**Significato**
`-a`	controlla tutti i filesystem montati.
`-R`	se specificata con `-a` non controlla la radice.
`-F`	controlla le quote nel formato specificato, prende i valori: `vfsold` (per la versione 1), `vfsv0` (per la versione 2), `rpc` (per le quote su NFS) e `xfs` (per le quote su XFS).
`-u`	richiede l'esecuzione del controllo per le sole quote degli utenti (il default).
`-g`	richiede l'esecuzione del controllo per le sole quote dei gruppi.
`-c`	non legge i file con i dati delle quote esistenti.
`-v`	riporta lo stato di avanzamento delle operazioni.
`-m`	forza il controllo delle quote senza rimontare il filesystem in sola lettura.
`-M`	esegue il controllo anche con il filesystem montato in scrittura se il tentativo di rimontarlo in sola lettura fallisce.

Tabella 6.13: Principali opzioni del comando `quotacheck`.

Il comando costruisce una tabella con l'uso corrente del disco e la confronta con i dati memorizzati su `aquota.user` e `aquota.group`, a meno di non usare l'opzione `-c`, nel qual caso detti file non vengono letti. Se trova delle incoerenze aggiorna le informazioni. In genere il comando necessita di montare il filesystem in sola lettura, onde evitare delle variazioni dello spazio occupato durante il calcolo; si può forzare l'esecuzione se il filesystem non è montabile in sola lettura usando l'opzione `-m`, nel qual caso il calcolo non sarà totalmente attendibile.

Di default il comando opera solo per le quote utente, se si vogliono controllare solo le quote gruppo occorre specificarlo esplicitamente usando l'opzione `-g`, se le si vogliono controllare entrambe va specificata esplicitamente anche l'opzione `-u`. Le altre opzioni principali sono riportate in tab. 6.13, per l'elenco completo ed i dettagli al solito si faccia riferimento alla pagina di manuale.

Una volta che si è montata una partizione con il controllo delle quote perché questo sia applicato è necessaria una attivazione esplicita con il comando `quotaon`. Come per `quotacheck`

il comando richiede come argomento uno (o più) file di dispositivo su cui è situato il filesystem con le quote o l'opzione `-a` per attivarli tutti. Il comando prende le opzioni `-u` e `-g` con lo stesso significato di `quotacheck` per attivare le quote su utenti e gruppi ed al solito di default sono attivate soltanto quelle sugli utenti. L'opzione `-v` aumenta la prolissità del comando stampando ogni filesystem per il quale si sono attivate le quote, mentre con `-p` si stampa solo lo stato delle stesse invece di attivarle. Per le altre opzioni ed i dettagli di funzionamento al solito si può fare riferimento alla pagina di manuale.

In sostanza per poter utilizzare le quote, una volta modificato `/etc/fstab` per indicare i filesystem su cui usarle, si dovrà provvedere a lanciare negli script di avvio (subito dopo il montaggio dei filesystem) un opportuno script che invochi prima `quotacheck` e poi `quotaon`. Gran parte delle distribuzioni installano automaticamente uno script di questo tipo con l'installazione dei programmi di controllo delle quote.[21]

Il sistema del controllo delle quote può anche essere disabilitato utilizzando `quotaoff`. Il comando (che è equivalente a chiamare `quotaon` con l'opzione `-f`) notifica al kernel la richiesta di non effettuare più controlli sul superamento delle quote. Il comando prende gli stessi argomenti di `quotaon` (in realtà si tratta dello stesso eseguibile, solo invocato con un nome diverso), e le due opzioni `-v` e `-p` hanno lo stesso effetto descritto in precedenza.

6.4.3 Gestione delle quote di utenti e gruppi

Una volta configurato il supporto delle quote, e attivato lo stesso con quanto illustrato nella sezione precedente, il passo successivo è quello di impostare quali restrizioni (sia in termini di spazio disco che di *inode* disponibili) si vogliono imporre ai vari utenti (o ai gruppi).

Come accennato in sez. 6.4.1 le quote si applicano in maniera indipendente per ciascun filesystem, inoltre se si ricorda quanto illustrato in sez. 5.1.4, un filesystem unix-like ha due tipi di risorse, i *blocchi* e gli *inode*, i primi sono relativi allo spazio disco utilizzabile, i secondi al numero di file, e su entrambi possono essere applicate delle restrizioni in maniera indipendente.

La struttura del meccanismo delle quote disco prevede la presenza di due limiti, uno *morbido* (detto appunto *soft limit*) che può essere superato per brevi periodi di tempo, ed uno *duro* (detto *hard limit*) che non può mai venir superato. Il breve periodo di tempo durante il quale è possibile superare il primo limite è detto *periodo di grazia* (*grace period*), durante detto periodo è ancora possibile creare nuovi file o usare spazio disco superando il limite *morbido*, ma comunque mai al di là del limite *duro*.

Opzione	**Significato**
`-f`	limita le impostazioni al filesystem specificato.
`-u`	richiede l'impostazione delle quote per un utente.
`-g`	richiede l'impostazione delle quote per un gruppo.
`-t`	imposta il *periodo di grazia*.
`-p`	specifica un utente di riferimento.

Tabella 6.14: Principali opzioni del comando `edquota`.

Il programma che permette di impostare le quote (ovviamente solo all'amministratore) è `edquota`, e nella sua forma più semplice prende come argomento l'utente di cui si vogliono

[21] nel caso di Debian lo script è `/etc/init.d/quota` e fa parte del pacchetto omonimo.

modificare le quote, il comando infatti per default agisce sulle quote utente, se si vuole operare su un gruppo occorre specificare l'opzione `-g`. Il comando crea un file di testo temporaneo con la rappresentazione delle quote correnti per l'utente o il gruppo scelto, e lancia l'editor di default (in genere `vi` o quello stabilito dalla variabile di ambiente `EDITOR`), all'interno del quale apparirà un contenuto del tipo:

```
Quotas for user piccardi:
/dev/hda3: blocks in use: 2594, limits (soft = 5000, hard = 6500)
         inodes in use: 356, limits (soft = 1000, hard = 1500)
```

le informazioni riguardo l'uso corrente sono riportate solo per riferimento, e una loro modifica verrà ignorata, modificando invece i valori dei limiti fra parentesi, salvando il file ed uscendo dall'editor questi verranno utilizzati come nuovi limiti. Un limite può essere annullato usando il valore 0.

In realtà il comando consente di specificare come argomenti più utenti (o gruppi) ed utilizzare l'opzione `-p` per specificare un utente (o gruppo) di riferimento le cui impostazioni dovranno essere utilizzate per tutti gli altri. In questo modo è possibile effettuare una impostazione generale per un singolo utente senza doverla poi ripetere a mano per tutti gli altri.

Se non si specifica nulla il comando imposta le quote per tutti i filesystem che le supportano; si può delimitare l'azione del programma ad un solo filesystem specificando quest'ultimo (indicato per file di dispositivo) con l'opzione `-f`.

Infine con l'opzione `-t` si può impostare il valore del cosiddetto *periodo di grazia*. Si tenga presente che questa è una proprietà globale che può essere impostata solo a livello di tutti gli utenti (o tutti i gruppi) di un singolo filesystem. Usando questa opzione di nuovo sarà aperto un file temporaneo all'interno dell'editor di default, il cui contenuto sarà qualcosa del tipo:

```
Time units may be: days, hours, minutes, or seconds
Grace period before enforcing soft limits for users:
/dev/hda2: block grace period: 0 days, file grace period: 0 days
```

Con la prima versione della gestione delle quote specificare un valore nullo per il *periodo di grazia* significava utilizzare il valore di default del periodo, pari ad una settimana. Con le nuove versioni è sempre necessario impostare esplicitamente un limite. Un elenco delle principali opzioni del comando `edquota` è riportato in tab. 6.14, per i dettagli e l'elenco completo si faccia al solito riferimento alla pagina di manuale.

Una volta attivate le quote ci sono due comandi che permettono di verificarne lo stato, il primo è `quota` che permette ad un utente di verificare le condizioni delle sue quote disco. Il comando con l'opzione `-u` (il default, può essere omessa) stampa un resoconto delle quote utente, se si vuole controllare le quote di gruppo occorre usare l'opzione `-g`.

L'amministratore (e solo lui) può anche richiedere il resoconto delle quote di un altro utente (o di un gruppo, nel qual caso occorre usare l'opzione `-g`) specificando quest'ultimo come argomento del comando. Le altre opzioni principali di `quota` sono riportate in tab. 6.15, per i dettagli e l'elenco completo si faccia al solito riferimento alla pagina di manuale.

Un secondo comando per il controllo dello stato delle quote è `repquota` che però è ad uso esclusivo dell'amministratore, in quanto serve a fornire dei rapporti globali sullo stato delle quote

Opzione	Significato
`-f`	limita le impostazioni al filesystem specificato.
`-u`	richiede i dati delle quote utente.
`-g`	richiede i dati delle quote gruppo.
`-l`	riporta solo le quote su filesystem locali.
`-q`	stampa le informazioni in modalità *succinta*.

Tabella 6.15: Principali opzioni del comando `quota`.

nel sistema riportando i totali per tutti gli utenti. Un esempio del suo risultato potrebbe essere il seguente:

```
root@monk# repquota -a
                        Block limits           File limits
        User         used    soft    hard  grace    used  soft  hard  grace
        root      --  175419      0       0          14679     0     0
        bin       --   18000      0       0            735     0     0
        uucp      --     729      0       0             23     0     0
        man       --      57      0       0             10     0     0
        user1     --   13046  15360   19200            806  1500  2250
        user2     --    2838   5120    6400            377  1000  1500
```

Il comando `repquota` prevede come argomento il filesystem di cui si vogliono controllare le quote specificato sia tramite *mount-point* che per dispositivo; questo può essere omesso se si usa l'opzione `-a` che controlla tutti i filesystem per cui sono abilitate le quote. Le altre opzioni servono per lo più a controllare le informazioni che vengono stampate; in particolare al solito `-u` e `-g` consentono di richiedere il rapporto rispettivamente per quote utenti e quote gruppo. Un sommario delle principali opzioni è riportato in tab. 6.16, al solito per l'elenco completo ed i dettagli di funzionamento del comando si può fare riferimento alla pagina di manuale.

Opzione	Significato
`-a`	elenca i dati per tutti i filesystem in cui `/etc/mtab` riporta l'uso di quote.
`-u`	elenca i dati delle quote utente.
`-g`	elenca i dati delle quote gruppo.
`-n`	non risolve i nomi di utenti e gruppi.
`-q`	stampa le informazioni in modalità *succinta*.
`-s`	adatta le unità di misura per spazio disco e *inode*.
`-v`	riporta maggiori informazioni e stampa i dati delle quote anche quando non sono usate.

Tabella 6.16: Principali opzioni del comando `repquota`.

La gestione delle quote attraverso `edquota` risulta però abbastanza scomoda, dovendo passare attraverso l'uso di un editor e modifiche di file di testo, il che ad esempio rende sostanzialmente inutilizzabile il comando all'interno di script. Per questo sono stati creati degli altri programmi che consentono di eseguire l'impostazione delle quote direttamente sulla riga di comando.

Il primo di questi è `setquota`. Usato nella sua forma elementare il comando prende come argomenti un nome utente (ma si può usare un gruppo se si specifica l'opzione `-g`) seguito dai limiti soft e hard per i blocchi e poi da quelli per gli *inode*. L'ultimo argomento è il filesystem a cui applicare le quote, che può essere omesso se si usa l'opzione `-a`. Con l'opzione `-b` i limiti delle

quote possono essere forniti sullo standard input invece che sulla riga di comando. Un sommario delle principali opzioni è riportato in tab. 6.17, al solito per l'elenco completo ed i dettagli di funzionamento del comando si può fare riferimento alla pagina di manuale.

Opzione	**Significato**
`-a`	applica il comando a tutti i filesystem in cui `/etc/mtab` riporta l'uso di quote.
`-u`	richiede l'impostazione delle quote utente (indicato per nome o *user ID*).
`-g`	richiede l'impostazione delle quote gruppo (indicato per nome o *group ID*).
`-b`	legge i dati delle quote dallo standard input invece che dagli argomenti della riga di comando.
`-t`	imposta il *periodo di grazia* (in secondi), richiede che si specifichino come argomenti il tempo in secondi per blocchi e *inode*.

Tabella 6.17: Principali opzioni del comando `setquota`.

Un secondo comando di gestione delle quote, che oltre alle impostazioni può anche eseguire la visualizzazione delle stesse, è `quotatool`. L'unico argomento preso dal comando è il filesystem su cui si vuole operare, che può essere indicato sia per dispositivo che per *mount-point*. Occorre inoltre indicare esplicitamente utente o gruppo su cui si vuole operare con le opzioni `-u` o `-g`, cui deve seguire come parametro il nome utente o il rispettivo *user ID*, a meno che non si sia usata anche l'opzione `-t` per impostare il periodo di grazia a livello globale.

Opzione	**Significato**
`-u`	imposta le quote utente.
`-g`	imposta le quote gruppo.
`-d`	stampa le quote correnti per l'utente o il gruppo.
`-t`	imposta il *periodo di grazia*, può essere usato un valore numerico con i prefissi `+` e `-` per aumentarlo o diminuirlo ed un suffisso che specifichi le unità di misura (`seconds`, `minutes`, ecc.)
`-i`	indica di modificare le quote per gli *inode*.
`-b`	indica di modificare le quote per i blocchi.
`-l`	imposta un *hard-limit*.
`-q`	imposta un *soft-limit*.

Tabella 6.18: Principali opzioni del comando `quotatool`.

Le opzioni `-i` e `-b` servono per indicare se si vogliono impostare le quote dei blocchi o degli *inode*, mentre i valori delle stesse devono essere specificati come parametri per le opzioni `-q` (per il *soft-limit*) e `-l` (per l'*hard-limit*). Le principali opzioni sono riportate in tab. 6.18, per l'elenco completo ed i dettagli di funzionamento al solito si può fare riferimento alla pagina di manuale.

Capitolo 7

Amministrazione di base delle reti

7.1 Un'introduzione ai concetti fondamentali delle reti.

Il campo della comunicazione via rete è uno dei più vasti e complessi di tutta l'informatica. Nel corso degli anni sono stati sviluppati molti mezzi (cavo, fibra, radio), e molti protocolli (TCP/IP, AppleTalk, IPX, ecc.) che permettessero di far comunicare fra loro computer diversi.

In generale con il termine di rete si identifica quella serie di meccanismi e metodi di collegamento tramite i quali tanti singoli computer, chiamati *nodi* o *stazioni* (in inglese *host*) vengono messi in comunicazione fra di loro in modo da potersi scambiare dati.

In questa sezione introduttiva esamineremo in breve alcuni concetti di base relativi alle reti, a partire dai diversi criteri che vengono usati come base per la loro classificazione, quale l'estensione, la *topologia* con cui sono realizzate ed i protocolli di comunicazione usati.

7.1.1 L'estensione

Una delle forme di classificazione di una rete più elementari, ed una delle meno precise, è quella per estensione o *area*. In origine, quando le reti erano disponibili solo nei grandi centri di ricerca o nelle grandi imprese, la classificazione era molto semplice e prevedeva soltanto le due tipologie:

LAN *Local Area Network*, che indica le reti locali, realizzate su brevi distanze, all'interno di uno stesso edificio o gruppo limitrofo di edifici (in genere una università o la sede di una grande impresa) che di norma sono possedute e gestite da una sola organizzazione.

WAN *Wide Area Network*, che indica in generale una rete di grande estensione, e a cui in genere si fa riferimento per indicare la struttura che connette varie reti locali, estendendosi potenzialmente su tutto il mondo (*Internet* è un esempio di WAN). In questo caso la rete non è proprietà di una entità singola, ma viene gestita in comune da più enti o organizzazioni.

Con il diffondersi dei computer e delle tecnologie di comunicazione, anche questa classificazione delle reti si è evoluta, e la suddivisione per area di estensione si è diversificata notevolmente, tanto che attualmente oltre alle due precedenti sono state introdotte una lunga serie di altre tipologie, come ad esempio:

MAN *Metropolitan Area Network*, che indica in genere una rete cittadina, di dimensione intermedia fra LAN e WAN, in cui varie reti locali vengono integrate preliminarmente fra di loro a livello di area metropolitana, grazie a delle infrastrutture dedicate. Viene di norma gestita da entità legate al governo della città.

SAN *Storage Area Network*, che fa riferimento alle reti dedicate a fornire un sistema di stoccaggio dati comune ad un insieme di computer. In generale sono usate all'interno di singole organizzazioni e non sono da considerarsi propriamente delle reti di comunicazione.

PAN *Personal Area Network*, che indica in genere quelle reti di comunicazione usate per connettere computer e dispositivi di uso personale (ad esempio il telefonino), su distanze di pochi metri, come il *bluetooth*.

DAN *Desktop Area Network*, che in genere fa riferimento alle reti di comunicazione usate per connettere vari dispositivi periferici ad un computer a brevissima distanza (la scrivania appunto) come le reti ad infrarossi IRDA.

In generale comunque, la sola classificazione rilevante è la prima, fra LAN e WAN. Le altre si sovrappongono spesso fra di loro (come per DAN e PAN) o non hanno a che fare, come la SAN (e le tecnologie di comunicazione usate per i cluster, o all'interno dei computer per le varie CPU) con quello che tratteremo in questo testo.

7.1.2 La topologia

Una classificazione generale applicabile a qualunque rete è quella basata sulla sua topologia. La topologia (dal greco *topos*) è una branca della geometria che studia le proprietà delle connessioni fra oggetti geometrici, e si applica pertanto anche alla struttura delle reti.

La classificazione riportata prevede solo alcune topologie di base, in genere poi le reti sono costruite con l'interconnessione di reti diverse e possono assumere delle topologie ibride rispetto a quelle qui elencate. Le topologie fondamentali sono:

point-to-point È la forma di connessione più semplice, ed in realtà in questo caso non si può neanche propriamente parlare di una rete; in quanto si realizza semplicemente una connessione diretta fra due macchine, che si vedono appunto come i due capi di una connessione punto-punto.

È la topologia classica delle connessioni via *modem*,[1] ma vale anche per altri tipi di collegamento fisico che consentono soltanto una trasmissione da un capo ad un altro (come quello tramite seriale o tramite porta parallela) in questo caso semplicemente si inviano i dati da una macchina verso l'altra e viceversa.

[1] con *modem* si intende fare riferimento appunto ad un apparato che consente di collegare, appoggiandosi ad una tecnologia di connessione qualunque, due macchine; il caso più comune, da cui deriva il nome (abbreviazione di *modulatore-demodulatore*) è quello classico dei modem analogici che usano le linee telefoniche.

bus È una rete che utilizza un canale centrale (detto *backbone*) per connettere fra loro tutti i dispositivi. Il canale funziona come collettore cui ogni stazione si collega con un connettore inviandovi tutti i messaggi. Ogni stazione riceve tutto il traffico del canale, estraendone i messaggi a lei indirizzati.

È la topologia classica delle vecchie reti Ethernet su BNC, fortunatamente ormai totalmente scomparse, ma si ritrova ad esempio nel caso delle connessioni tramite USB. In genere questo tipo di rete ha il vantaggio della facilità di installazione e di bassi costi per gli apparati, che non devono gestire politiche di distribuzione dei dati, ma è efficace solo per un numero limitato di stazioni, in quanto il canale deve distribuire ad ogni nodo tutto il traffico, compreso quello destinato agli altri, e può essere facilmente saturato. Inoltre in caso di rottura del canale centrale tutto il traffico di rete è totalmente bloccato e l'intera rete non è usabile.

ring È una rete in cui ogni stazione ha due stazioni vicine, a loro volta collegate ad un'altra stazione, fino a formare un anello. Ciascuna stazione comunica con le stazioni limitrofe, ed i messaggi vengono inoltrati lungo l'anello per essere ricevuti dalla stazione di destinazione. È potenzialmente più efficiente della struttura a *bus* in quanto non è necessario che il messaggio attraversi tutto l'anello, ma solo la parte necessaria a raggiungere la destinazione. Anche in questo caso però la rottura di un cavo o di una stazione comporta l'inutilizzabilità dell'intera rete.

Questa topologia di rete, un tempo era diffusa anche nella realizzazione di reti locali grazie alla tecnologia denominata *token-ring*, sviluppata e promossa dalla IBM. In questo ambito è oggi sostituita in maniera quasi completa dalle reti ethernet, ma viene invece utilizzata in maniera abbastanza estesa per i collegamenti in fibra ottica.

star È una rete in cui esiste uno snodo centrale, un *hub* o uno *switch*,[2] cui vengono connesse le varie stazioni. Richiede normalmente una maggiore estensione della cablatura, ma la rottura di un cavo non interrompe tutta la rete. Inoltre usando uno *switch* i pacchetti vengono smistati direttamente dalla stazione di origine alla destinazione, e non si soffre del problema del collo di bottiglia dovuto alla necessità di trasmettere tutti i dati ad ogni stazione collegata.

Questa è la topologia utilizzata dalle versioni recenti delle reti ethernet, ed è probabilmente quella più comune nella realizzazione di reti locali.

tree È una rete che integra in forma gerarchica più reti a stella. Viene in genere realizzata connettendo su una *backbone* o su uno switch centrale diversi *hub* o *switch* periferici. In questo modo si possono aumentare le stazioni collegate superando i limiti sul numero di dispositivi collegati ad un singolo switch e limitare la quantità di traffico che deve passare per la *backbone*.

mesh È una rete che comporta la presenza di diversi percorsi possibili per la comunicazione. È in genere la modalità in cui vengono create le reti WAN, o i cosiddetti *fabric switch*, in cui, per ottimizzare la velocità di trasmissione, si effettuano connessioni incrociate

[2]in realtà un *hub* appartiene a questa topologia solo dal punto di vista delle connessioni fisiche, dato che poi tutto il traffico viene comunque reinviato a tutte le stazioni connesse.

fra più *switch*, così da avere strade diverse per il flusso dei dati, con un traffico più distribuito, che permette di comunicare con latenze inferiori.

7.1.3 I protocolli

Come abbiamo accennato parlare di reti significa parlare di un insieme molto vasto ed eterogeneo di mezzi di comunicazione che vanno dal cavo telefonico, alla fibra ottica, alle comunicazioni via satellite o via radio. Per rendere possibile la comunicazione attraverso un così variegato insieme di mezzi sono stati adottati una serie di protocolli, il più famoso dei quali, quello alla base del funzionamento di Internet, è il protocollo TCP/IP.

Una caratteristica comune dei protocolli di rete è il loro essere strutturati in livelli sovrapposti. In questo modo ogni protocollo di un certo livello può realizzare le sue funzionalità basandosi su un protocollo del livello sottostante. Questo modello di funzionamento è stato standardizzato dalla *International Standards Organization* (ISO) che ha preparato fin dal 1984 il Modello di Riferimento *Open Systems Interconnection* (OSI), strutturato in sette livelli, secondo quanto riportato in tab. 7.1.

Livello	**Nome**	
Livello 7	*Application*	*Applicazione*
Livello 6	*Presentation*	*Presentazione*
Livello 5	*Session*	*Sessione*
Livello 4	*Transport*	*Trasporto*
Livello 3	*Network*	*Rete*
Livello 2	*DataLink*	*Collegamento Dati*
Livello 1	*Physical*	*Connessione Fisica*

Tabella 7.1: I sette livelli del protocollo ISO/OSI.

La definizione di ciascuno di questi livelli è la seguente:

Applicazione
Il livello di applicazione indica il livello finale in cui un utente interagisce con l'applicazione. Si indicano come esempi di questo livello le applicazioni per l'uso di telnet e FTP.

Presentazione
Il livello di presentazione fornisce le funzionalità di codifica e conversione dei dati usate dal successivo livello di applicazione, come la rappresentazione dei caratteri, i formati dei dati (audio, video, immagini), gli schemi di compressione, la cifratura.

Sessione
Il livello di sessione gestisce, crea e termina sessioni di comunicazione; è a questo livello che sono impostati, definiti e sincronizzati gli scambi di dati. Di norma è qui che sono definiti i protocolli di funzionamento dei singoli servizi (telnet, FTP, e-mail) che vengono offerti sulla rete.

Trasporto
Il livello di trasporto fornisce la comunicazione tra le applicazioni che girano sulle due stazioni terminali; è sostanzialmente equivalente all'omonimo livello del modello TCP/IP illustrato più avanti.

Rete Il livello di rete si occupa dello smistamento dei singoli pacchetti su una rete interconnessa, ed è a questo livello che si definiscono gli indirizzi di rete che identificano macchine non in diretto collegamento fisico; è sostanzialmente equivalente all'omonimo livello del modello TCP/IP illustrato più avanti.

Collegamento Dati
Il livello di collegamento dati è quello che fornisce una trasmissione affidabile dei dati su una connessione fisica. A questo livello si definiscono caratteristiche come l'indirizzamento dei dispositivi, la topologia della rete, la sequenza dei pacchetti, il controllo del flusso e la notifica degli errori sul livello di connessione fisica (cioè delle singole stazioni collegate direttamente fra loro da una connessione fisica, e non attraverso i livelli superiori del protocollo).

La IEEE ha diviso questo livello in due ulteriori parti, il *Logical Link Control* (LLC), definito nello standard IEEE 802.2, che gestisce le comunicazioni fra dispositivi all'interno di un singolo collegamento in una rete, ed il *Media Access Control* (MAC) che gestisce l'accesso al mezzo fisico, e consente a dispositivi diversi di identificarsi univocamente in una rete: a questo livello sono definiti gli indirizzi fisici, detti appunto *MAC address*, delle schede di rete (che tratteremo in sez. 7.2.1).

Connessione fisica
Il livello di connessione fisica si occupa delle caratteristiche materiali (elettriche, fisiche, meccaniche) e funzionali per attivare, disattivare e mantenere il collegamento fisico fra diverse reti di comunicazione. Sono definiti a questo livello caratteristiche come l'ampiezza e la temporizzazione dei segnali, il tipo dei connettori, la massima capacità di trasmissione, le distanze raggiungibili.

Il modello ISO/OSI è stato sviluppato in corrispondenza alla definizione della serie di protocolli X.25 per la commutazione di pacchetto. Nel frattempo però era stato sviluppato un altro protocollo di comunicazione, il TCP/IP, su cui si basa Internet, che è diventato uno standard de facto. Il modello di quest'ultimo, più semplice, viene chiamato anche modello DoD (sigla che sta per *Department of Defense*), dato che fu sviluppato dall'agenzia ARPA per conto del Dipartimento della Difesa Americano.

Così come ISO/OSI anche il modello del TCP/IP è stato strutturato in livelli (riassunti in tab. 7.2). Un confronto fra i due modelli è riportato in fig. 7.1, dove viene evidenziata anche la corrispondenza fra i rispettivi livelli, che comunque è approssimativa. Si è indicato in figura anche fra quali livelli, nel sistema operativo, viene inserita l'interfaccia di programmazione (i cosiddetti *socket*) per l'accesso alla rete.

Livello	**Nome**		**Esempi**
Livello 4	*Application*	*Applicazione*	Telnet, FTP, etc.
Livello 3	*Transport*	*Trasporto*	TCP, UDP
Livello 2	*Network*	*Rete*	IP, (ICMP, IGMP)
Livello 1	*Link*	*Collegamento*	ethernet, PPP, *Wi-Fi*

Tabella 7.2: I quattro livelli del protocollo TCP/IP.

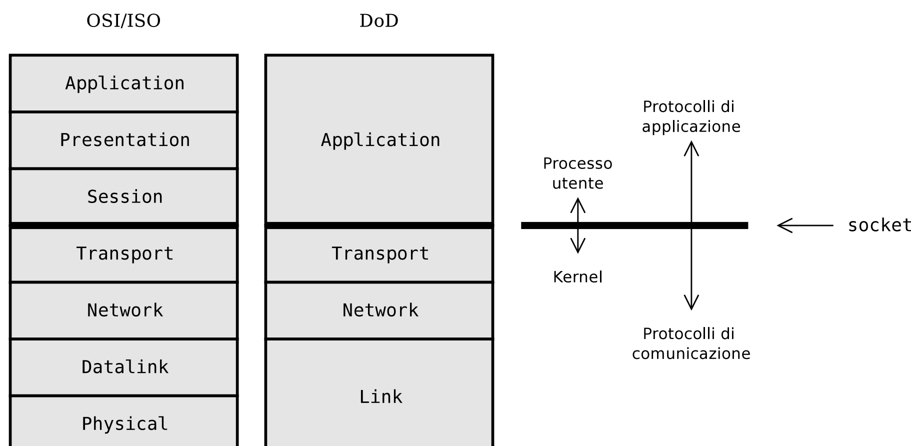


Figura 7.1: Confronto fra il modello OSI ed il modello TCP/IP nella loro relazione con il sistema.

Il nome del modello deriva dai due principali protocolli su cui è basata Internet, il TCP (*Transmission Control Protocol*) che copre il livello 3, e l'IP (*Internet Protocol*) che copre il livello 2. Le funzioni dei vari livelli sono le seguenti:

Applicazione È relativo ai programmi di interfaccia con la rete, in genere questi vengono realizzati secondo il modello *client-server*,[3] realizzando una comunicazione secondo un protocollo che è specifico di ciascuna applicazione.

Trasporto Fornisce la comunicazione tra singole applicazioni che girano sulle stazioni terminali, regola il flusso delle informazioni, può fornire un trasporto affidabile, cioè con recupero degli errori o inaffidabile. I protocolli principali di questo livello sono il TCP e l'UDP.

Rete Si occupa dello smistamento dei singoli pacchetti su una rete complessa e interconnessa, a questo stesso livello operano i protocolli per il reperimento delle informazioni necessarie allo smistamento, per lo scambio di messaggi di controllo e per il monitoraggio della rete. Il protocollo su cui si basa questo livello è IP (sia nella attuale versione, IPv4, che nella nuova versione, IPv6).

Collegamento È responsabile per l'interfacciamento al dispositivo elettronico che effettua la comunicazione fisica, gestendo l'invio e la ricezione dei pacchetti da e verso l'hardware.

Quale dei due modelli usare è spesso una questione di gusti; il modello ISO/OSI è più teorico e generale, il modello TCP/IP è più legato alla struttura con cui la gestione della rete è implementata in un sistema GNU/Linux. Per questo motivo, e data la sua maggiore semplicità, nel resto del testo faremo riferimento solo a quest'ultimo.

[3]si intende con questo una modalità di comunicazione in cui un programma di richiesta, detto *client*, interroga opportunamente un programma di servizio, detto *server*.

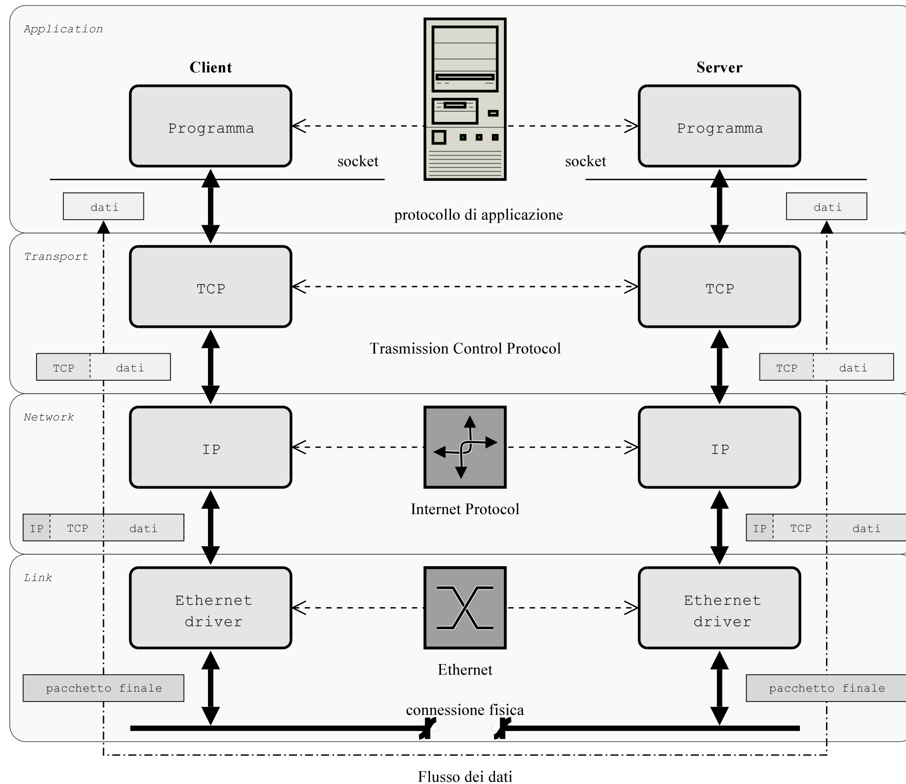


Figura 7.2: Strutturazione del flusso dei dati nella comunicazione fra due applicazioni attraverso i protocolli della suite TCP/IP.

Per cercare di capire meglio le ragioni di tutta questa suddivisione in livelli consideriamo un'analogia con quanto avviene nella spedizione di una lettera per posta aerea in America. Voi scrivete la vostra bella lettera su un foglio, che mettete in una busta con l'indirizzo che poi imbucate. Il postino la raccoglierà dalla buca delle lettere per portarla al centro di smistamento, dove in base alla sua destinazione sarà impacchettata insieme a tutte quelle che devono essere inviate per posta aerea, e mandata al successivo centro di raccolta. Qui sarà reimpacchettata insieme a quelle provenienti dagli altri centri di smistamento ed imbarcata sull'aereo. Una volta scaricate in America le varie lettere saranno spacchettate e reimpacchettate per lo smistamento verso la destinazione finale. Qui un altro postino prenderà la vostra e la metterà nella cassetta della posta del vostro destinatario, il quale la aprirà e leggerà quello che gli avete scritto.

Questo è molto simile a quello che succede quando volete spedire dei dati via rete, secondo il procedimento che è illustrato in fig. 7.2. Ad ogni passaggio attraverso un livello del protocollo al nostro pacchetto di dati viene aggiunta una intestazione relativa al protocollo utilizzato in quel livello, si dice cioè che un pacchetto di un protocollo viene "*imbustato*" nel protocollo successivo. A differenza della posta in questo caso i pacchetti con i dati non vengono disfatti ad ogni livello per passare in un pacchetto diverso, ma vengono reimbustati pari pari al livello successivo, i vari

"*pacchetti*" vengono disfatti (e la relativa intestazione rimossa) solo quando si torna indietro ad un livello superiore.

Questo meccanismo fa si che il pacchetto ricevuto ad un livello n dalla stazione di destinazione sia esattamente lo stesso spedito dal livello n dalla stazione sorgente. Tutto ciò rende facile progettare il software in maniera modulare facendo riferimento unicamente a quanto necessario ad un singolo livello, con la confidenza che questo poi sarà trattato uniformemente da tutti i nodi della rete.

7.2 Il TCP/IP.

Come accennato in sez. 7.1.3 negli anni sono stati creati molti tipi diversi di protocolli per la comunicazione via rete, in questo testo però prenderemo in esame soltanto il caso di reti basate su TCP/IP, il "*protocollo*"[4] più diffuso, quello su cui si basa "*Internet*", e che ormai è diventato uno standard universale e disponibile su qualunque sistema operativo. In questa sezione ci limiteremo ad introdurre i concetti fondamentali delle reti IP, la notazione e le terminologie principali.

7.2.1 Introduzione.

Dato che il protocollo è nato su macchine Unix, il TCP/IP è la modalità di comunicazione nativa di GNU/Linux, e benché per compatibilità siano stati implementati nel kernel anche parecchi altri protocolli di comunicazione (come DECnet, AppleTalk, IPX), questo resta il principale ed il più usato e non soltanto in ambiente Unix.

Come illustrato in sez. 7.1.3 l'insieme di protocolli che costituisce il TCP/IP è strutturato, secondo l'omonimo modello, su 4 livelli; a ciascuno di essi corrisponde un particolare compito, svolto da uno (o più) protocolli specifici. In fig. 7.3 si è riportata una panoramica dei principali protocolli che costituiscono il TCP/IP e come questi vengono suddivisi nei quattro livelli illustrati in precedenza.

L'interfaccia fondamentale usata dal sistema operativo per la comunicazione in rete è quella dei cosiddetti *socket*, nata nel 1983 a Berkeley e resa pubblica con il rilascio di BSD 4.2. La flessibilità e la genericità dell'interfaccia consente di utilizzare i socket con i più disparati meccanismi di comunicazione, e non solo con l'insieme dei protocolli TCP/IP, si ricordi infatti, come illustrato in sez. 1.2.1, che essi sono disponibili anche come oggetti sul filesystem (i cosiddetti *socket locali*).

Con i socket infatti si può creare un canale di comunicazione fra due stazioni remote, sul quale inviare i dati direttamente, senza doversi preoccupare di tutto il procedimento di passaggio da un livello all'altro che viene eseguito dal kernel. Tutte le applicazioni che forniscono i principali servizi su Internet (pagine WEB, posta elettronica, connessione remota) sono realizzate a livello di applicazione usando questa interfaccia. Solo per applicazioni specialistiche per il controllo della rete il kernel mette a disposizione delle ulteriori interfacce che permettono di accedere direttamente ai livelli inferiori del protocollo.

[4] in realtà non si tratta di un solo protocollo, ma di una *suite* in cui sono inseriti vari protocolli, in modo da coprire in maniera sostanzialmente completa le più varie esigenze di comunicazione.

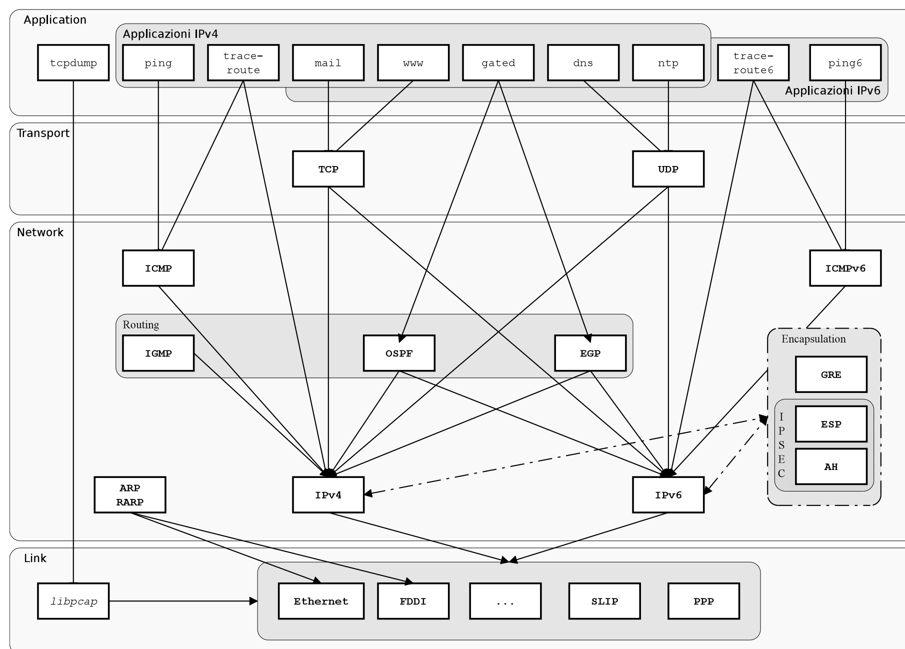


Figura 7.3: Panoramica sui vari protocolli che compongono la suite TCP/IP.

Come mostrato in fig. 7.2 i *socket* fanno da ponte fra il livello di applicazione e quello di trasporto; una volta inviati su un socket i dati vengono passati (dal kernel) ad uno dei protocolli del livello di trasporto, che sono quelli che si curano di trasmettere i dati dall'origine alla destinazione, secondo le modalità specificate dal tipo di socket scelto. Così se si è usato uno *stream socket* basato sul TCP quest'ultimo si curerà di stabilire la connessione con la macchina remota e garantire l'affidabilità della comunicazione controllando eventuali errori, ritrasmettendo i pacchetti persi e scartando quelli duplicati.[5]

Al livello successivo sarà poi IP che curerà l'invio dei singoli pacchetti sulla rete, ed è su questo livello che operano i vari dispositivi che su Internet consentono lo smistamento dei pacchetti fino alla loro destinazione finale. È a questo livello che sono definiti gli indirizzi IP che identificano ogni macchina sulla rete.

Il livello finale è quello dell'interfaccia di rete usata per trasmettere i pacchetti verso l'esterno, che a seconda dei casi può essere una scheda Ethernet o l'interfaccia associata al modem; questo si incaricherà di trasmettere i dati sulla rete fisica che connette la macchina all'esterno (sia questa una rete locale o Internet) utilizzando anche qui gli opportuni protocolli di comunicazione.

Benché quest'ultimo livello non faccia strettamente parte del protocollo TCP/IP, è comunque importante approfondire brevemente le caratteristiche dei protocolli di comunicazioni fisica usati sulle reti locali di tipo *broadcast*, che fanno riferimento alla famiglia di IEEE 802.x, e che coprono

[5]in questo caso dal punto di vista dell'utente la trasmissione dei dati è più simile ad un collegamento telefonico che ad un invio di lettere; useremo questa ed altre analogie nel seguito, ma si deve essere consapevoli che nessuna di esse è mai perfettamente congruente con il comportamento effettivo della rete.

le più diffuse tecnologie come appunto Ethernet dato che queste hanno comunque importanti riflessi anche sulle caratteristiche del TCP/IP. Esamineremo invece le connessioni punto-punto in sez. 7.5.

Come accennato in sez. 7.1.3 lo standard IEEE 802.2 definisce una modalità generica di comunicazione che fa ricorso a degli indirizzi fisici denominati *MAC address*. Questi indirizzi sono realizzati come numeri a 48 bit,[6] e derivano dalle specifiche originali del protocollo Ethernet, nonostante il loro utilizzo sia stato adottato da moltissime altre tecnologie.

La notazione prevede che vengano espressi nella forma di 6 ottetti scritti in cifre esadecimali separate dal carattere "`:`" (o alternativamente dal "`-`"). I primi tre ottetti indicano il produttore di una scheda, sono assegnati dalla IEEE e vengono anche chiamati OUI (*Organizationally Unique Identifier*) i successivi tre ottetti sono a scelta del produttore e possono essere qualunque, ma devono essere univoci per ogni scheda prodotta. Ad esempio `00:19:99:58:dc:30` è il *MAC address* di una scheda della Fujitsu.

Una ulteriore convenzione è che il secondo bit meno significativo del primo ottetto viene utilizzato, se 0, per indicare che l'indirizzo è fra quelli utilizzati dai produttori, che vengono chiamati *universally administered*. Se invece è 1 si tratta di indirizzi detti *locally administered*, che possono essere impostati dall'amministratore di rete; ad esempio `00:0e:a6:f6:0a:6d` è l'indirizzo di una scheda di rete virtuale usata per una macchina virtuale.

I *MAC address* vengono usati per identificare la scheda che ha inviato un pacchetto (con il *MAC address* sorgente), e quella che lo deve ricevere (con il *MAC address* destinazione) e con questi viene gestita la trasmissione dei pacchetti nella rete locale. In genere una scheda legge solo i pacchetti destinati al suo *MAC address*; inoltre l'indirizzo (di destinazione) `FF:FF:FF:FF:FF:FF` è l'indirizzo di *broadcast*, che deve essere ricevuto da tutte le schede.

In fig. 7.3 si sono riportati i protocolli principali che costituiscono il TCP/IP. In questo testo copriremo solo quelli di utilità più comune; una breve descrizione di quelli riportati in figura è comunque la seguente:

IPv4 *Internet Protocol version 4*. È quello che comunemente si chiama IP. Ha origine negli anni '80 e da allora è la base su cui è costruita Internet. Usa indirizzi a 32 bit, e mantiene tutte le informazioni di instradamento e controllo per la trasmissione dei pacchetti sulla rete; tutti gli altri protocolli della suite (eccetto ARP e RARP, e quelli specifici di IPv6) vengono trasmessi attraverso di esso.

IPv6 *Internet Protocol version 6*. È stato progettato a metà degli anni '90 per rimpiazzare IPv4. Ha uno spazio di indirizzi ampliato a 128 bit che consente più gerarchie di indirizzi, l'autoconfigurazione, ed un nuovo tipo di indirizzi, gli *anycast*, che consentono di inviare un pacchetto ad una stazione su un certo gruppo. Effettua lo stesso servizio di trasmissione dei pacchetti di IPv4 di cui vuole essere un sostituto ed una evoluzione.

TCP *Transmission Control Protocol*. È un protocollo di trasporto orientato alla connessione che fornisce un trasporto affidabile per un flusso di dati bidirezionale fra due stazioni remote. Il protocollo ha cura di tutti gli aspetti del trasporto, come l'ack-

[6] la notazione ufficiale li definisce come *MAC-48*, oppure *EFI-48*, esiste una estensione a 64 bit chiamata *EFI-64*.

noweledgment, i timeout, la ritrasmissione, ecc. È usato dalla maggior parte delle applicazioni.

UDP *User Datagram Protocol.* È un protocollo di trasporto senza connessione, per l'invio di dati a pacchetti. Contrariamente al TCP il protocollo non è affidabile e non c'è garanzia che i pacchetti raggiungano la loro destinazione, si perdano, vengano duplicati, o abbiano un particolare ordine di arrivo.

ICMP *Internet Control Message Protocol.* È il protocollo usato a livello di rete per gestire gli errori e trasportare le informazioni di controllo fra *stazioni remote* e *instradatori* (cioè fra *host* e *router*). I messaggi sono normalmente generati dal software del kernel che gestisce la comunicazione TCP/IP, anche se ICMP può venire usato direttamente da alcuni programmi come `ping`. A volte ci si riferisce ad esso come ICMPv4 per distinguerlo da ICMPv6, la versione usata con IPv6.

IGMP *Internet Group Management Protocol.* È un protocollo di livello di rete usato per il *multicasting*, cioè la trasmissione uno a molti. Permette alle stazioni remote di notificare ai *router* che supportano questa comunicazione a quale gruppo esse appartengono. Come ICMP viene implementato direttamente sopra IP.

DHCP *Dynamic Host Configuration Protocol.* È un protocollo (vedi sez. 8.2.1) che permette di inviare informazioni di configurazione alle stazioni presenti in una rete locale, come indirizzo IP, indirizzi di *gateway* e server DNS, file per l'avvio via rete, ecc.

ARP *Address Resolution Protocol.* È il protocollo che mappa un indirizzo IP in un *MAC address* sulla rete locale. Viene usato per le reti di tipo *broadcast* (come Ethernet, Token Ring o FDDI). Torneremo su di esso in sez. 7.6.4. Non serve in connessioni punto-punto.

RARP *Reverse Address Resolution Protocol.* È il protocollo che esegue l'operazione inversa rispetto ad ARP (da cui il nome) mappando un indirizzo hardware in un indirizzo IP. Veniva usato per assegnare un indirizzo IP ad una macchina al suo avvio, ma è stato sostituito da altri protocolli specifici dedicati a questo (vedi sez. 8.2.1).

ICMPv6 *Internet Control Message Protocol, version 6.* Combina per IPv6 le funzionalità dei protocolli ICMPv4, IGMP e ARP usati con IPv4.

EGP *Exterior Gateway Protocol.* È un protocollo di instradamento usato per comunicare lo stato fra *gateway* vicini a livello di *sistemi autonomi*[7], con meccanismi che permettono di identificare i vicini, controllarne la raggiungibilità e scambiare informazioni sullo stato della rete. Viene implementato direttamente sopra IP.

OSPF *Open Shortest Path First.* È un protocollo di instradamento per *router* su reti interne, che permette a questi ultimi di scambiarsi informazioni sullo stato delle connessioni e dei legami che ciascuno ha con gli altri. Viene implementato direttamente sopra IP.

[7] vengono chiamati *autonomous systems* i raggruppamenti al livello più alto della rete.

GRE *Generic Routing Encapsulation.* È un protocollo generico di incapsulamento che permette di incapsulare un qualunque altro protocollo all'interno di IP.

AH *Authentication Header.* Fornisce l'autenticazione dell'integrità e dell'origine di un pacchetto IP. È una opzione nativa in IPv6 e viene implementato come protocollo a sé su IPv4. Fa parte della suite di IPSEC che provvede la trasmissione cifrata ed autenticata a livello IP (vedi sez. 4.1.2 di [SGL]).

ESP *Encapsulating Security Payload.* Provvede la cifratura insieme all'autenticazione dell'integrità e dell'origine di un pacchetto IP. Come per AH è opzione nativa in IPv6 e viene implementato come protocollo a sé su IPv4. Anch'esso fa parte di IPSEC (vedi sez. 4.1.2 di [SGL]).

PPP *Point-to-Point Protocol.* È un protocollo di livello di collegamento progettato per lo scambio di pacchetti su connessioni punto-punto. Viene usato per configurare i collegamenti, definire i protocolli di rete usati ed incapsulare i pacchetti di dati. È un protocollo complesso con varie componenti (vedi sez. 7.5).

SLIP *Serial Line over IP.* È un protocollo di livello di collegamento che permette di trasmettere un pacchetto IP attraverso una linea seriale.

7.2.2 Gli indirizzi IP (IPv4 e IPv6)

Per poter comunicare fra loro due computer in rete devono potersi in qualche modo riconoscere. Gli indirizzi IP, definiti dall'Internet Protocol visto in sez. 7.1.3, svolgono esattamente questo ruolo, che è analogo a quello del numero di telefono o dell'indirizzo di una casa. Essi devono identificare univocamente un nodo della rete. Dato che a tutt'oggi la versione più utilizzata del protocollo è IPv4, faremo nel seguito riferimento ad indirizzo IP intendendo l'indirizzo previsto da questo protocollo, tratteremo brevemente gli indirizzi IPv6, indicandoli esplicitamente come tali, nella parte finale di questa sezione.

L'analogia più diretta è quella con il telefono; per poter telefonare occorre avere un proprio numero di telefono e conoscere quello di colui che si vuole chiamare. L'indirizzo IP è l'equivalente del numero di telefono, solo che invece che di un numero decimale composto di un numero variabile di cifre è un numero binario ed ha una dimensione fissa di quattro byte. Come per i telefoni ad un numero può corrispondere solo un telefono, ma normalmente ad un telefono non possono essere associati più numeri, vedremo più avanti invece come ad uno stesso computer (o a una stessa interfaccia di rete) possono essere assegnati più indirizzi IP.

Di solito, visto che scrivere i numeri in formato binario è poco comprensibile, si è soliti esprimere il numero che costituisce l'indirizzo IP usando una apposita notazione che viene chiamata *dotted decimal*, usando un numero decimale per ciascuno dei quattro byte che compongono l'indirizzo, separati da un punto; un esempio di questa notazione potrebbe essere qualcosa del tipo di `192.168.111.11`. È attraverso questo numero che il vostro computer viene identificato univocamente su Internet.

Come per il telefono di casa ogni computer connesso ad Internet viene sempre considerato come facente parte di una *rete*; la cosa è vera anche per le reti private non connesse direttamente

ad Internet (come quelle che collegano i computer di un ufficio). Per proseguire nell'analogia si pensi alle linee telefoniche interne di una ditta, usate per parlare tra uffici interni.

Dato che Internet, come dice il nome, è un insieme di reti, anche queste devono venire identificate. Questo è fatto attraverso altrettanti indirizzi IP, che corrispondono alla parte *comune* di tutti gli indirizzi delle macchine sulla stessa rete. Per proseguire nell'analogia con il telefono si può pensare all'indirizzo di rete come al prefisso che serve per parlare con un altra città o un altro stato, e che è lo stesso per tutti i telefoni di quell'area.

La differenza con i prefissi è che un indirizzo di rete IP, quando lo si scrive, deve essere completato da tanti zeri quanti sono necessari a raggiungere la dimensione di 32 bit degli indirizzi normali; per riprendere il precedente esempio di indirizzo IP, un possibile indirizzo di rete ad esso relativo potrebbe essere `192.168.111.0`.

Questo meccanismo significa in realtà che ogni indirizzo su Internet, pur essendo espresso sempre come un singolo numero, nei fatti è composto da due parti, *l'indirizzo di rete*, che prende la parte superiore dell'indirizzo e identifica la particolare sezione di Internet su cui si trova la vostra rete e *l'indirizzo della stazione* (il cosiddetto *host*), che prende la parte inferiore del numero e che identifica la macchina all'interno della vostra rete.

La situazione dunque è ancora analoga a quella di un numero di telefono che è diviso in prefisso e numero locale. Un prefisso è l'equivalente dell'indirizzo di rete, l'indirizzo IP completo è quello che identifica il singolo telefono, solo che in questo caso il numero di cifre (binarie) che si usano per il *prefisso* non è fisso, e può anche essere cambiato a seconda dei casi.

Per questo motivo, quando si configura una macchina, ad ogni indirizzo IP si associa sempre anche quella che viene chiamata una *netmask*: una maschera binaria che permette di dire quali bit dell'indirizzo sono usati per identificare la rete e quali per il nodo. Questa viene espressa con la solita notazione *dotted decimal*, usando il numero binario costruito mettendo un 1 ad ogni bit dell'indirizzo corrispondente alla rete e uno zero a quello corrispondente alla stazione: nel caso dell'indirizzo in esempio si avrebbe allora una *netmask* uguale a `255.255.255.0`.

L'assegnazione degli indirizzi IP è gestita a livello internazionale dalla IANA (*Internet Assigned Number Authority*), che ha delegato la gestione di parte delle assegnazioni ad altre organizzazioni regionali (come INTERNIC, RIPE NCC e APNIC). Originariamente, per venire incontro alle diverse esigenze, gli indirizzi di rete erano stati organizzati in *classi*, (riportate tab. 7.3), per consentire dispiegamenti di reti di dimensioni diverse.

Classe	**Intervallo**	**Netmask**
A	`0.0.0.0` — `127.255.255.255`	`255.0.0.0`
B	`128.0.0.0` — `191.255.255.255`	`255.255.0.0`
C	`192.0.0.0` — `223.255.255.255`	`255.255.255.0`
D	`224.0.0.0` — `239.255.255.255`	`240.0.0.0`
E	`240.0.0.0` — `247.255.255.255`	

Tabella 7.3: Le classi di indirizzi IP.

Le classi usate per il dispiegamento delle reti di cui è attualmente composta Internet sono le prime tre; la classe D è destinata al cosiddetto *multicast*,[8] che è una modalità di comunicazione

[8]per poter utilizzare una comunicazione in *multicast* una macchina deve registrarsi presso il *router* per segnalare appunto a quale *gruppo di multicast* intende aderire; in questo modo il *router* potrà reinviarle i pacchetti di *multicast* ricevuti per esso.

per consentire la spedizione simultanea di uno stesso pacchetto IP da una stazione singola, che lo emette una volta sola, verso più stazioni riceventi. Il *multicast* permette di ottimizzare trasmissione dati da uno verso molti e viene usato per distribuire contenuti; per utilizzare questa modalità di comunicazione occorre operare su uno di questi indirizzi. Infine la classe E è riservata per usi sperimentali e non viene impiegata.

Oggigiorno questa divisione in classi non è più molto usata (perché come vedremo fra poco è inefficiente), ma la si trova riportata spesso, ed alcuni programmi cercano di calcolare automaticamente la *netmask* a seconda dell'IP che gli date, seguendo queste tabelle, e questo alle volte può creare problemi, dato che non è detto che la rete in questione sia ancora classificabile in questo modo.

Una rete di classe A è una rete che comprende 16777216 (cioè 2^{24}) indirizzi di singoli computer ed ha una netmask pari a `255.0.0.0`, una rete di classe B comprende 65536 (cioè 2^{16}) indirizzi ed ha una netmask pari a `255.255.0.0` e una rete di classe C comprende 256 (cioè 2^{8}) indirizzi ed ha una netmask pari a `255.255.255.0`.

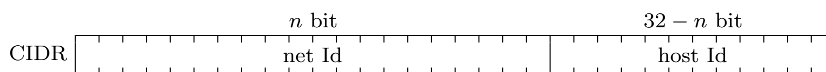


Tabella 7.4: Uno esempio di indirizzamento CIDR.

La suddivisione riportata in tab. 7.3 è largamente inefficiente in quanto se un utente necessita di anche solo un indirizzo in più dei 256 disponibili[9] con una classe C occorre passare a una classe B, con un conseguente enorme spreco di numeri (si passerebbe da 256 a 65536).

Per questo nel 1992 è stato introdotto un indirizzamento senza classi (detto CIDR) in cui il limite fra i bit destinati a indicare l'indirizzo di rete e quelli destinati a indicare l'host finale può essere piazzato in qualunque punto dei 32 bit totali (vedi tab. 7.4), permettendo così di accorpare più classi C su un'unica rete o suddividere una classe B. È stata così introdotta anche una nuova notazione, che permette di indicare la parte di rete appendendo all'indirizzo l'indicazione del numero di bit riservati alla rete; nel caso in esempio si avrebbe allora `192.168.111.11/24`.

Oltre alla comunicazione fra due singoli nodi della rete, quella descritta finora con l'analogia telefonica, su Internet è possibile un'altra modalità di comunicazione, detta *broadcast*. In questo caso è un singolo nodo che può inviare pacchetti che saranno ricevuti da tutti gli altri nodi presenti nella sua stessa rete; questa modalità di comunicazione è analoga alla trasmissione via radio, da cui deriva il nome (in inglese).

Per permettere questo tipo di comunicazione su ogni rete, oltre agli indirizzi dei singoli nodi ed a quello della rete stessa, è necessario disporre di un indirizzo riservato, che viene chiamato appunto indirizzo di *broadcast*. Questo per convenzione è sempre ottenuto mettendo ad 1 tutti i bit della porzione dell'indirizzo IP riservata all'host. È questo indirizzo che deve essere utilizzato da una singola stazione quando vuole inviare un messaggio contemporaneamente a tutti gli altri nodi presenti su quella rete. Questo permette di usare direttamente le capacità di alcune tecnologie di rete, come la Ethernet, che supportano questa modalità di comunicazione.

[9] in realtà gli indirizzi disponibili con una classe C sono 254, questo perché l'indirizzo `.0` è riservato per indicare la rete, mentre l'indirizzo `.255`, come vedremo fra poco, è quello di *broadcast*; questi indirizzi sono riservati per ogni rete e non possono essere usati per un singolo host.

La presenza di un indirizzo di *broadcast* permette il funzionamento di una serie di protocolli ausiliari del TCP/IP che devono poter scambiare e ricevere informazioni con tutti i computer presenti (ad esempio quelli che permettono di scoprire quali sono gli indirizzi realmente attivi), senza doversi indirizzare a ciascuno di essi individualmente.

Indirizzo	**Esempio**
Indirizzo completo	`192.168.111.11`
Maschera di rete	`255.255.255.0`
Porzione di rete	`192.168.111.`
Porzione del nodo	`.11`
Indirizzo di rete	`192.168.111.0`
Indirizzo broadcast	`192.168.111.255`

Tabella 7.5: Specchietto riassuntivo della struttura degli indirizzi IP.

In tab. 7.5 si è riassunta la struttura generica di un indirizzo IP all'interno di una rete, con tutte le tipologie di indirizzi, ed i vari valori numerici che possono essere richiesti nella configurazione di una interfaccia di rete per l'uso di un indirizzo IP.

Nella assegnazione di un indirizzo IP occorre tenere conto che alcuni indirizzi sono riservati e non possono essere utilizzati su Internet. Ad esempio dalla classe A è stata rimossa l'intera rete `127.0.0.0` che viene associata alla speciale interfaccia di rete detta *loopback*.[10] Si tratta di una interfaccia virtuale che effettua la comunicazione localmente facendo passare i pacchetti attraverso il kernel per farli comparire su se stessa (da cui il suo nome); è convenzione seguita universalmente associare a questa interfaccia l'indirizzo `127.0.0.1`, detto *localhost*, che viene utilizzato per comunicare, senza usare fisicamente la rete, quando un programma che usa i socket TCP/IP deve fare riferimento ad un altro programma che gira sulla stessa macchina.

Un altro indirizzo speciale è l'indirizzo nullo, `0.0.0.0`, che viene usato per indicare un indirizzo generico non specificato. Lo si usa anche per indicare l'insieme di tutti gli indirizzi possibili,[11] e per questo viene anche usato per indicare la cosiddetta *default route*, cioè la destinazione verso cui inviare tutti i pacchetti con una destinazione al di fuori della rete locale.

Classe	**Intervallo**
A	`10.0.0.0` — `10.255.255.255`
B	`172.16.0.0` — `172.31.255.255`
C	`192.168.0.0` — `192.168.255.255`

Tabella 7.6: Le classi di indirizzi IP riservate per le reti private.

Infine l'RFC 1918 riserva una serie di indirizzi per le cosiddette *reti private*, cioè per le reti interne ad una organizzazione che non devono mai essere connesse direttamente ad Internet. Si tratta di una rete di classe A, 16 reti di classe B e 256 reti di classe C; i loro indirizzi sono riportati in tab. 7.6.

Originariamente si intendeva per *reti private* proprio reti che non dovevano essere interconnesse fra loro, ad uso esclusivo di comunicazione interna. Oggi buona parte di queste reti possono

[10] è quella indicata dalla sigla `lo`, interna al singolo nodo, che serve per spedire pacchetti a se stessi.

[11] ed in effetti se lo si prende come indirizzo di rete corrisponde appunto alla rete che ha come host tutti i possibili indirizzi, e cioè Internet.

collegarsi verso Internet, attraverso l'uso del cosiddetto NAT (*Network Address Translation*) che consente di far uscire intere reti IP passando con un singolo indirizzo IP pubblico,[12] dove si ha un accesso dall'interno verso Internet, e non viceversa.

Negli anni '90 con la crescita del numero di macchine connesse su Internet, e prima dello sviluppo delle tecnologie di *Network Address Translation*, si arrivò a temere l'esaurimento dello spazio degli indirizzi disponibili, specie in vista di una prospettiva (poi rivelatasi prematura) in cui ogni apparecchio elettronico sarebbe stato inserito all'interno della rete. Per questo motivo si iniziò a progettare una nuova versione del protocollo IP, che poi, con un salto di versione, portò ad IPv6.

Dal punto di vista più superficiale il problema deriva dal fatto che con 32 bit si hanno 2^{32}, cioè "*solo*" 4 miliardi circa, numeri diversi possibili. In realtà questi ancora oggi sono di più degli apparati effettivamente collegati in rete, ma la suddivisione di questi numeri con l'utilizzo delle classi di indirizzamento di tab. 7.3, ha comportato che nella sua evoluzione storica il dispiegamento delle reti e l'allocazione degli indirizzi siano stati inefficienti, e neanche l'uso del CIDR ha permesso di eliminare le inefficienze che si erano formate.

Per molti anni la riallocazione delle reti e l'uso del NAT hanno reso meno urgente il problema ma oggi, la crescita dei dispositivi mobili connessi ad internet, e lo sviluppo delle tecnologie che consentono di costruire computer di dimensioni microscopiche dotati di connessione, ha iniziato a far sentire davvero il limite sul numero di indirizzi IPv4 disponibili: l'assegnazione delle ultime reti IPv4 rimaste disponibili è stata effettuata dalla IANA il 3 febbraio 2011.

Per risolvere questi problemi il protocollo IPv6 è stato sviluppato come evoluzione di IPv4, mantenendone inalterate le funzioni che si sono dimostrate valide, eliminando quelle inutili e aggiungendone poche altre ponendo al contempo una grande attenzione a mantenere il protocollo il più snello e veloce possibile. I cambiamenti apportati sono comunque notevoli e possono essere riassunti a grandi linee nei seguenti punti:

- l'espansione delle capacità di indirizzamento e instradamento, per supportare una gerarchia con più livelli di indirizzamento, un numero di nodi indirizzabili molto maggiore e una auto-configurazione degli indirizzi;
- l'introduzione un nuovo tipo di indirizzamento, l'*anycast* che si aggiunge agli usuali *unicast* e *multicast*;
- la semplificazione del formato dell'intestazione dei pacchetti, eliminando o rendendo opzionali alcuni dei campi di IPv4;
- un supporto per le opzioni migliorato, per garantire una trasmissione più efficiente del traffico normale, limiti meno stringenti sulle dimensioni delle opzioni, e la flessibilità necessaria per introdurne di nuove in futuro;
- il supporto per delle capacità di qualità di servizio (QoS) che permetta di identificare gruppi di dati per i quali si può provvedere un trattamento speciale (in vista dell'uso di internet per applicazioni multimediali e/o "real-time").

Il cambiamento più importate resta comunque il primo, che ha visto passare il numero di bit usati per indicare un indirizzo IPv6 da 32 a 128, cosa che dovrebbe risolvere il problema dell'esaurimento se non in maniera definitiva sicuramente in una prospettiva a lunghissimo termine; con questi numeri infatti si hanno a disposizione svariate centinaia di migliaia di miliardi

[12] l'argomento è trattato in sez. 3.2 di [SGL].

di indirizzi per ogni metro quadro di superficie terrestre. Ma anche se questo è un numero completamente teorico, che deve scontrarsi con eventuali inefficienze di allocazione, si è comunque giunti alla conclusione che anche nella peggiore delle ipotesi IPv6 dovrebbe essere in grado di fornire più di un migliaio di indirizzi per ogni metro quadro della superficie terrestre.

Dato che il numero di bit è quadruplicato, non è più possibile usare la notazione coi numeri decimali di IPv4 per rappresentare un indirizzo IP. Per questo gli indirizzi di IPv6 sono in genere scritti come sequenze di otto sezioni di numeri esadecimali di quattro cifre cadauno (indicante un gruppo di 16 bit) usando i due punti come separatore. Avremo cioè con un indirizzo del tipo: `2001:0db8:0000:0000:0008:ba98:2078:e3e3`.

Visto che la notazione resta comunque piuttosto pesante esistono alcune abbreviazioni; si può evitare di scrivere gli zeri iniziali delle singole sezioni per cui l'indirizzo precedente si potrebbe abbreviare in `2001:db8:0:0:8:ba98:2078:e3e3`. Se poi una intera sezione è nulla la si può omettere del tutto, così come un insieme di più sezioni nulle (ma questo solo per quella più a sinistra, per non generare ambiguità). Per questo motivo il precedente indirizzo si potrebbe scrivere anche come `2001:db8::8:ba98:2078:e3e3`.

Come per IPv4 la parte iniziale dell'indirizzo indica una rete, che in questo caso deve essere espressa unicamente in notazione "CIDR", indicando il numero di bit che copre. Inoltre IPv6 supporta esplicitamente l'indicazione contemporanea di un indirizzo e della sua rete scrivendo quest'ultimo nella forma `2001:db8::8:ba98:2078:e3e3/64`.

Come per IPv4 esistono alcuni indirizzi riservati, ad esempio in IPv6 il *localhost* si esprime come `::1/128` (cioè tutti zeri con un 1 finale). Gli indirizzi nella classe `fe80::/10` sono detti *link-local* ed indicano indirizzi che sono validi solo all'interno di un link fisico diretto su una rete locale per far parlare fra loro le macchine di una LAN. Si possono considerare una sorta di analogo degli IP delle reti private di tab. 7.6,[13] anche se il loro significato è diverso e vengono usati per assegnare automaticamente alla propria scheda di rete un indirizzo IPv6. In realtà fra questi sono stati allocati soltanto quelli della sottoclasse `fe80::/64`, che vengono usati per creare un indirizzo IPv6 direttamente dal *MAC address* di una scheda di rete.

Gli indirizzi IPv4 vengono poi mappati su IPv6 con una classe di indirizzi specifica nella forma `::ffff:0:0/96` (detti *IPv4 mapped address* in cui ci sono 80 bit a zero, seguiti da 16 bit a uno e dai 32 bit dell'indirizzo IPv4. Per questi vale la ulteriore notazione di poter esprimere le ultime due cifre di un indirizzo IPv6 in notazione dotted decimal, pertanto l'indirizzo IPv4 `192.168.1.1` verrebbe espresso come `::ffff:192.168.1.1`. Una forma alternativa, chiamata *IPv4 compatible address* prevede soltanto zeri iniziali usa cioè la classe `::0/96`, ed in tal caso il precedente indirizzo diverrebbe `::192.168.1.1`.

7.2.3 L'instradamento o *routing*

L'identificazione di un nodo sulla rete effettuata tramite l'indirizzo IP è solo il primo passo per poter stabilire effettivamente una comunicazione. Una volta che si sappia la destinazione finale infatti, occorre sapere anche come poterci arrivare. Il lavoro di smistamento e reindirizzamento verso la loro destinazione finale dei pacchetti di dati che vengono trasmessi via rete è quello che in termini tecnici viene chiamato *instradamento*, in inglese *routing*.

[13] in realtà i veri equivalenti sarebbero quelli della classe `fec0::/10`, ma la creazione di reti locali in questa modalità non ha senso con IPv6 e questa assegnazione è stata deprecata.

Il *routing* è uno degli argomenti più complessi delle tecnologie di rete, ma qui lo tratteremo soltanto nei suoi aspetti più elementari. In generale il problema si può dividere in due: nella parte relativa alla propria rete locale, ed in quello che succede una volta usciti da essa. Il funzionamento del *routing* per la trasmissione dei pacchetti su Internet comporta una serie di problematiche estremamente complesse, per cui ci limiteremo ad una semplice descrizione funzionale, ma dal punto di vista di una rete locale tutto si riduce al problema di come fare arrivare all'*esterno* i pacchetti che escono dai singoli computer in essa presenti.

In questo caso si può ancora fare riferimento all'analogia telefonica: si può pensare alla propria rete locale come alla rete telefonica interna di una ditta; per poter uscire e telefonare all'esterno occorre in qualche modo passare dal centralino. Su Internet le macchine che fanno da "centralino" e permettono il collegamento di reti diverse smistando fra di esse i pacchetti loro destinati sono dette *router*.

In una rete locale il ruolo del centralino è svolto dal cosiddetto *default gateway*, questa è la macchina che nella vostra rete fa da ponte verso l'esterno, ed in generale indica la macchina che vi consente di raggiungere Internet. Tutte le telefonate dirette fuori (cioè tutti i pacchetti di dati che devono uscire dalla rete locale per andare su Internet) devono passare da questa macchina; per questo quando installate una macchina in una rete locale dovete sempre sapere l'indirizzo del *default gateway*.

La differenza coi numeri telefonici sta però nel fatto che l'indicazione di usare un centralino non si può inserire all'interno del numero che si compone (ad esempio mettendo un 0 o un 1 all'inizio dello stesso), ma in questo caso deve essere proprio specificato il numero completo della macchina che fa da ponte, che anch'essa avrà un suo indirizzo IP. Un'altra differenza è che con il TCP/IP quando si deve far arrivare un pacchetto da una rete ad un altra, anche se non si dovesse uscire su Internet (ad esempio due reti interne diverse), deve essere sempre utilizzato un *gateway*, cioè una macchina che possa fare da "*cancello di ingresso*" per i pacchetti destinati a quella rete.

Una volta usciti dalla rete locale la situazione si complica molto, ed in questo caso l'analogia telefonica non ci aiuta, perché il collegamento telefonico nasce a *commutazione di linea*. Per realizzarlo cioè si mettevano effettivamente in collegamento elettrico (la cosiddetta *continuità galvanica*) i due telefoni con una serie di selettori. Internet invece è progettata per funzionare a *commutazione di pacchetto*: i dati inviati vengono spezzati in pacchetti e passati da un nodo della rete all'altro fino ad arrivare alla loro destinazione finale, dove vengono riassemblati. Non esiste cioè una connessione diretta fra due nodi, anche se poi i programmi usano delle funzionalità che permettono di lavorare come se le cose fossero così.

Per questo, come dice poi lo stesso nome di *instradamento*, un'analogia che illustra più chiaramente i concetti del *routing* è quella delle reti stradali. Ad esempio quando lavoravo per l'INFN mi capitava spesso di dover andare al CERN. Per farlo prendevo l'autostrada a Firenze Sud, a Firenze Nord cambiavo sulla Firenze Mare, uscendo a Lucca per fare il raccordo per prendere l'autostrada per Genova, da Genova proseguivo per Alessandria, cambiavo di nuovo per Torino, dove fatta la circonvallazione prendevo l'autostrada per il traforo del Monte Bianco. Da lì il raccordo porta sull'autostrada per Ginevra, nella cui periferia sono i laboratori del CERN.

Come si può notare si tratta di un bel percorso complicato, che comporta il passaggio da diversi svincoli, incroci e caselli autostradali. Quando si invia un pacchetto su Internet succede qualcosa di molto simile, e anche lui deve passare attraverso degli analoghi dei "*caselli*". Quello che succede ad esempio quando ci si collega con un modem e si inizia a navigare un sito web,

è che i pacchetti che escono dal proprio computer vengono inviati al *router* del vostro provider che vi fa da *default gateway* (l'equivalente del casello). Da lì prenderanno la strada opportuna per arrivare al *router* del provider a cui è collegato il server web del sito, che li manderà a quest'ultimo.

In tutto questo percorso i pacchetti passeranno per una serie di altri *router* che sanno che strada devono prendere i pacchetti per poter arrivare alla destinazione finale. La differenza fra i caselli o svincoli stradali ed i *router* è che questi ultimi sanno indicare da soli ai pacchetti la strada su cui devono andare per arrivare a destinazione. In realtà sono ancora più intelligenti, e sono in grado di far prendere ai pacchetti la strada più veloce, tenendo conto di eventuali ingorghi, incidenti, interruzioni del traffico ecc. Così se il tunnel del Monte Bianco viene chiuso, quando si arriva a Torino il "*router*" farà deviare automaticamente verso il Frejus.

7.2.4 I servizi e le porte.

Finora abbiamo parlato quasi esclusivamente di indirizzi IP, ma come accennato nell'introduzione (si ricordi quanto detto in sez. 7.1.3) *Internet Protocol* è solo uno dei protocolli di Internet, e copre soltanto il *livello di rete*. Abbiamo già visto che per poter effettuare delle comunicazioni fra loro i programmi devono poter creare delle connessioni, e per far questo si deve usare l'interfaccia dei socket, che sono basati invece sui protocolli del *livello di trasporto*, come TCP ed UDP.

Occorre perciò introdurre un'altra delle caratteristiche del protocollo TCP/IP, relativa stavolta al *livello di trasporto*, senza comprendere la quale mancherebbero le basi per poter spiegare il funzionamento di quest'ultimo: quella delle *porte*. Anche in questo caso l'analogia telefonica ci viene, sia pure in maniera molto parziale, in aiuto. Finora infatti abbiamo parlato degli indirizzi IP come se fossero dei numeri di telefono, ma questo riguarda solo la parte del protocollo che viene usato per effettuare la trasmissione fra due computer, e cioè il protocollo IP.

Allora come su un numero di telefono può rispondere una persona (se solleva la cornetta), una segreteria telefonica, un fax, o un altro computer (se c'è attaccato un modem), lo stesso accade anche per Internet. Su un indirizzo IP possono cioè rispondere diversi *servizi*, corrispondenti a forme di comunicazione diversa, ed occorre identificare quello a cui ci si vuole rivolgere.

L'analogia telefonica risulta in questo caso molto debole perché di solito per usufruire dei servizi citati ci vogliono apparecchi diversi, anche se talvolta si trovano oggetti che assommano più di uno di essi. Per questo un'altra analogia più appropriata potrebbe essere quella di pensare alle *porte* come ai canali della filodiffusione,[14] cioè a delle specie di "*frequenze*" diverse su cui sintonizzate il vostro telefono, sulle quali trovate i contenuti più diversi.

Ma anche questa analogia è molto parziale, perché nel caso della filodiffusione il segnale non viene da un altro telefono, ma dal fornitore del servizio telefonico, e si può solo ascoltare, ed un canale alla volta, mentre con Internet si può sia ascoltare che trasmettere, da e verso qualunque altro "corrispondente", utilizzando in contemporanea un numero arbitrario di canali.[15]

Tutto questo è possibile perché, come spiegato in sez. 7.2.1, il TCP/IP è un insieme di protocolli, ed IP (quello degli indirizzi) è solo la parte che serve a gestire la trasmissione dei pacchetti da una macchina all'altra. Per poter effettuare uno scambio di dati fra due programmi

[14] per chi non ha idea di che cosa sia, si tratta di una specie di radio via telefono, usata per trasmettere musica quando le radio avevano una pessima qualità, ma che oggi non esiste praticamente più.

[15] in realtà a voler essere pignoli, il numero di canali contemporanei per singolo corrispondente può essere al massimo 65535 (pari a $2^{16} - 1$) e non totalmente arbitrario.

che comunicano via rete sono disponibili gli altri protocolli, che sono quelli che consentono di stabilire un canale di comunicazione diretto fra una applicazione ad un'altra.

È per questo motivo che nei protocolli del livello di *trasporto*, come UDP e TCP, è stato introdotto il concetto delle *porte*, cioè un numero ulteriore (oltre a quello dell'indirizzo) che permetta di identificare le due applicazioni che stanno usando il protocollo ai due capi della comunicazione. In questo modo diventa possibile gestire più connessioni contemporanee tra le stesse macchine (destinate a servizi diversi o provenienti da applicazione diverse) e tenere separati i pacchetti ad esse relative. In realtà poi per UDP sarebbe più corretto parlare di canali di comunicazione, non essendoci in senso stretto una connessione vera e propria come per il TCP, il quale, oltre all'invio dei pacchetti, permette anche di assicurare una comunicazione affidabile.

Questo significa che quando si vuole inviare della posta elettronica si comunicherà con la macchina che fornisce il servizio di spedizione utilizzando una certa porta, mentre quando si vuole leggere una pagina web se ne userà un'altra, diversa dalla precedente, effettuando con i servizi che si contattano degli scambi di dati specifici di ciascuno di essi, che vanno a costituire l'ultimo livello della struttura mostrata in fig. 7.1, quello di *applicazione*.

Si tenga conto però che in questa forma il concetto di porta può essere fuorviante, in quanto con porta si rischia di intendere una qualche forma di accesso permanente che può essere aperto o chiuso. In realtà non esiste nessuna forma di accesso permanente e lo scambio di dati avviene solo se si hanno da ambo le parti (lato server e lato client) gli strumenti per effettuarlo.

Questo significa che non sarà mai possibile sfondare una "*porta*" sul vostro computer, se su di essa non c'è un programma di servizio a ricevere i dati, così come non possono spaccarvi i timpani urlando alla radio, se è spenta. Per questo forse sarebbe più chiaro parlare di frequenza, su cui si può trasmettere o ascoltare, ed in cui ciascuno può essere la trasmittente (il server) o il ricevente (il client) o anche entrambi allo stesso tempo (ad esempio nei sistemi *peer to peer*).

Bussando ad una porta (o sintonizzandosi su quella frequenza a seconda dell'analogia che si preferisce) si potranno scambiare, attraverso l'opportuno protocollo di applicazione, i dati relativi al servizio associato. Dal punto di vista del TCP/IP si potrebbe usare un numero di porta qualsiasi, ma la standardizzazione ha portato ad associare alcuni numeri di porta a dei servizi specifici: la porta 25 alla posta elettronica, la porta 80 al web, ecc.

Si tenga presente inoltre che come c'è una porta per indicare quale servizio si vuole raggiungere sulla macchina di destinazione, ci deve essere anche una porta che identifichi il programma che ha fatto la richiesta sulla macchina di origine. Questo perché se ad esempio si lanciassero due istanze di un browser sulla propria macchina andando a leggere lo stesso sito, si avrebbe una situazione in cui si contatta un server web a partire da uno stesso indirizzo IP di origine, andando verso lo stesso indirizzo IP e la stessa porta di destinazione. Ma il proprio kernel deve essere in grado di poter separare i pacchetti di risposta alle due istanze del browser (che potrebbero leggere pagine diverse) e per questo le porte non esistono soltanto per i servizi di destinazione ma anche per i programmi che li contattano.

Questo significa che nella comunicazione con i protocolli del livello di trasporto (sia con TCP che con UDP) ogni client invierà i suoi pacchetti di richiesta e riceverà i pacchetti di risposta dal server utilizzando anch'esso una sua porta, che viene chiamata *porta sorgente*. Quella che viene usata per identificare un servizio che si vuole contattare, di cui si è parlato finora e che in genere è quella che si intende quando si parla semplicemente di porta, viene invece chiamata *porta destinazione*.

A differenza di quanto avviene con la porta destinazione, i cui numeri sono standardizzati, in genere la porta sorgente viene assegnata automaticamente dal kernel quando un programma crea una connessione. Per questo motivo viene chiamata anche *porta effimera* (o *ephemeral port*) proprio in quanto non necessita di avere un valore predeterminato, ed il suo utilizzo è limitato alla durata di una connessione.

Porta	**Protocollo**	**Servizio**
`20`	TCP	FTP (*File Transfer Protocol*) dati.
`21`	TCP	FTP (*File Transfer Protocol*) comandi.
`22`	TCP	SSH (*Secure Shell*).
`23`	TCP	Telnet.
`25`	TCP	SMTP (*Simple Mail Transport Protocol*).
`53`	UDP	DNS (*Domain Name Service*).
`67`	UDP	DHCP (*Dynamic Host Configuration Protocol*) client.
`68`	UDP	DHCP (*Dynamic Host Configuration Protocol*) server.
`80`	TCP	HTTP (*Hyper Test Transfer Protocol*).
`110`	TCP	POP3 (*Post Office Protocol* v.3).
`111`	TCP/UDP	RPC (*Remote procedure call*).
`123`	UDP	NTP (*Network Time Protocol*).
`143`	TCP	IMAP (*Internet Message Access Protocol*).
`161`	UDP	SNMP (*Simple Network Management*).
`389`	TCP	LDAP (*Lightweight Directory Access Protocol*).
`443`	TCP	HTTPs (HTTP su SSL).
`631`	TCP	IPP (*Internet Printing Protocol*).
`993`	TCP	IMAPs (IMAP su SSL).
`995`	TCP	POP3s (POP3 su SSL).

Tabella 7.7: I numeri di porta dei principali servizi.

In un sistema Unix le prime 1024 porte (di destinazione) sono dette *riservate* in quanto solo l'amministratore può mettervi in ascolto dei servizi. La corrispondenza fra queste porte ed i servizi che ci devono essere installati è regolata a livello internazionale: nessuno vi obbliga a rispettare la convenzione, ma se mettete la posta elettronica sulla porta 80 e il web sulla 25 avrete certamente delle grosse difficoltà a comunicare con gli altri, dato che in genere i browser cercano i siti sulla porta 80, la posta viene inviata usando la porta 25.

Al di sopra della porta 1024 qualunque utente può mettere un suo servizio, avviando un demone che si metta in ascolto sulla porta che preferisce. Alcune di queste però sono state usate tradizionalmente da servizi molto diffusi, per cui in certi casi si potrebbero avere dei conflitti. Come vedremo in sez. 7.4.4 c'è un elenco che associa i numeri di porta ai servizi che li utilizzano, che viene mantenuto nel file `/etc/services`, si sono riportati i valori associati ai servizi più rilevanti in tab. 7.7.

7.3 La configurazione di base

La configurazione della rete è una materia alquanto complessa e di una vastità impressionante; in particolare essendo numerosissimi, e spesso molto complessi, i servizi di rete, sono altrettanto numerose e complesse le configurazioni che si possono affrontare. In questa sezione però affronteremo solo la configurazione di base della rete, e non dei vari servizi che possono essere realizzati su di essa.

In particolare vedremo come effettuare le varie impostazioni relative all'uso e alla gestione dei tre livelli più bassi del protocollo TCP/IP, che sono la base su cui è costruito tutto il resto, ed esamineremo sia i comandi tradizionali con i quali si effettua la configurazione manuale della rete[16], che i file di configurazione usati nel procedimento di avvio per la configurazione automatica delle interfacce.

7.3.1 L'assegnazione degli indirizzi ed il comando `ifconfig`

Il primo passo per poter utilizzare la rete è quello di assegnare ad una interfaccia di rete il suo indirizzo IP. Il comando che consente di fare questo è `ifconfig`, che permette anche di impostare le varie caratteristiche delle interfacce di rete. Quello che serve nella maggior parte dei casi sono soltanto le opzioni che permettono di attivare e disattivare una interfaccia.

Se usato senza opzioni e senza specificare una interfaccia il comando mostra lo stato di tutte le interfacce attive. Questo è il primo passo da fare sempre prima di qualunque configurazione per vedere lo stato del sistema, ed anche dopo per controllare che sia tutto a posto. Un risultato possibile è il seguente:

```
root@havnor:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:19:99:0e:67:ae
          inet addr:192.168.0.234  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::219:99ff:fe0e:67ae/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:117369393 errors:0 dropped:0 overruns:0 frame:0
          TX packets:101518535 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2812934074 (2.6 GiB)  TX bytes:1134332202 (1.0 GiB)
          Interrupt:16 Base address:0x6000

eth0:0    Link encap:Ethernet  HWaddr 00:19:99:0e:67:ae
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:16

eth1      Link encap:Ethernet  HWaddr 00:90:27:2c:19:e4
          inet addr:192.168.168.1  Bcast:192.168.168.255  Mask:255.255.255.0
          inet6 addr: fe80::290:27ff:fe2c:19e4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1301606994 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2505164903 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1274648067 (1.1 GiB)  TX bytes:3870531783 (3.6 GiB)
          Interrupt:17 Memory:c0020000-c0040000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:10226970 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10226970 errors:0 dropped:0 overruns:0 carrier:0
```

[16] i comandi `ifconfig`, `route`, `arp` infatti sono da anni dichiarati deprecati, in favore del comando generico `ip` (vedi sez. 7.3.4) che consente sia di sostituirli che di effettuare le configurazioni delle funzionalità più avanzate del kernel; nonostante questo il loro utilizzo per le operazioni di base resta estremamente comune.

```
collisions:0 txqueuelen:0
RX bytes:1385547296 (1.2 GiB)  TX bytes:1385547296 (1.2 GiB)
```

Se si specifica come argomento il nome di una interfaccia il comando mostra solo lo stato dell'interfaccia specificata, se invece si vuole lo stato di tutte le interfacce esistenti sulla propria macchina comprese quelle non attive, che normalmente non vengono mostrate anche se presenti, occorre specificare l'opzione `-a`.

Nell'esempio precedente si è illustrata la situazione di una macchina su cui risultano 3 interfacce di rete. Due di esse (`eth0` e `eth1`) corrispondono a due schede di rete Ethernet, la terza (`lo`) è una interfaccia logica, la cosiddetta interfaccia di *loopback*, che viene usata per le comunicazioni locali, e che deve essere sempre attivata anche per i computer non connessi in rete. In questo caso tutte le interfacce fisiche sono di tipo *broadcast*, manca nell'esempio un caso, che troveremo più avanti in sez. 7.5, di una interfaccia impostata per una connessione *punto-punto*.

Valore	**Tipo indirizzo**
`UP`	l'interfaccia è attiva.
`NOARP`	il protocollo ARP (vedi sez. 7.6.4) non è attivo sull'interfaccia.
`RUNNING`	l'interfaccia sta funzionando.
`MULTICAST`	sull'interfaccia è attivato il *multicast*.
`BROADCAST`	sull'interfaccia è attivato il *broadcast*.
`POINTOPOINT`	l'interfaccia è in modalità punto-punto.
`LOOPBACK`	l'interfaccia è in modalità *loopback*.
`MTU`	*Maximum Transfer Unit* dell'interfaccia.[17]

Tabella 7.8: Stati riportati nella terza riga del comando `ifconfig` e relativo significato.

Si può notare come il comando riporti le varie caratteristiche delle singole interfacce: nella prima riga viene scritto il tipo di collegamento fisico, e, se presente per quel tipo di interfaccia, anche l'indirizzo fisico. Dalla seconda riga viene riportato il tipo ed il valore dell'indirizzo associato all'interfaccia, nel caso sono presenti sia un indirizzo IPv4 ordinario che l'indirizzo IPv6 *link-local* ottenuto dall'indirizzo fisico della scheda di rete. A questi segue una riga che indica lo stato corrente della stessa, in cui sono elencate una serie di valori il cui elenco e relativo significato è riportato in tab. 7.8. Infine sulle righe seguenti vengono stampate una serie di altre informazioni statistiche relative al traffico sostenuto dall'interfaccia.

Valore	**Tipo indirizzo**
`inet`	Indirizzo IPv4.
`inet6`	Indirizzo IPv6.
`ax25`	Indirizzo AX25.
`ddp`	Indirizzo AppleTalk.
`ipx`	Indirizzo Novell IPX.
`netrom`	Indirizzo AMPR Packet radio.

Tabella 7.9: Valori del parametro opzionale per indicare il tipo di indirizzo con `ifconfig`.

A parte il caso in cui si legge la configurazione generica, il comando richiede sempre come primo argomento il nome dell'interfaccia su cui operare; quando si intende impostare un indirizzo

[17] la *Maximum Transfer Unit* di una interfaccia di rete è la massima dimensione che un pacchetto può avere per poter essere trasmesso sulla rete su cui si affaccia la suddetta interfaccia, questa dipende dalla tecnologia sottostante, ed in genere per una interfaccia Ethernet vale 1500 byte; torneremo su questo in sez. 7.6.2.

sulla stessa è possibile specificarne il tipo con un argomento opzionale da specificare prima del relativo valore. I valori possibili per questo argomento sono riportati in tab. 7.9 ed il valore di default, sottinteso quando non si specifica nulla, è `inet`, che indica il protocollo IPv4.

Normalmente questo argomento non viene mai specificato in quanto il caso più comune di utilizzo del comando `ifconfig` è appunto quello dell'impostazione di un indirizzo IP. Da qui in avanti inoltre, se non specificato altrimenti, intenderemo sempre IPv4 tutte le volte che si parlerà di IP in maniera generica.

Il risultato del comando riportato a pag. 505 ci mostra anche che l'interfaccia `eth0` è *multihomed*; su di essa cioè sono presenti due indirizzi diversi, riportati nelle due sezioni separate introdotte dalle stringhe `eth0` ed `eth0:0`. Con `ifconfig` l'assegnazione di ulteriori indirizzi su una qualunque interfaccia di rete (il cosiddetto *IP aliasing*) si deve fare utilizzando delle interfacce "*virtuali*" nella forma `eth0:`*`N`* dove *`N`* è un numero intero.[18] Si noti comunque come le statistiche siano riportate solo per la prima istanza, in quanto esse non possono che essere uniche per ciascuna interfaccia.

Prima di configurare una interfaccia è opportuno verificare che essa non sia già attiva, perché quando si assegna un indirizzo IP un valore precedente sarà sovrascritto. Il comando `ifconfig` *`iface`* (dove *`iface`* è il nome della singola interfaccia che si vuole controllare, ad esempio `eth0`) ci mostrerà lo stato dell'interfaccia, dando un errore nel caso il supporto nel kernel non sia attivato o l'interfaccia non esista. Qualora invece si voglia disattivare una interfaccia attiva si potrà usare il comando:

```
ifconfig eth0 down
```

specificando l'opzione `down`, mentre per riattivarla, mantenendo le impostazioni che c'erano in precedenza, si potrà utilizzare il comando:

```
ifconfig eth0 up
```

Quando invece si deve assegnare un indirizzo la sintassi dipende dal tipo di collegamento, nell'esempio precedente abbiamo tutte interfacce relative a reti di tipo *broadcast* che si affacciano su una rete locale ed in questo caso si potrà assegnare un indirizzo ad una interfaccia ed allo stesso tempo attivarla con il comando:

```
ifconfig eth0 192.168.1.100
```

in cui è sottintesa l'opzione `up` poiché si dà per scontato che assegnare un indirizzo IP ad una interfaccia implichi anche la volontà di utilizzarla.

Si ricordi però che in questo caso ad ogni indirizzo IP è sempre associata una rete IP, che nel comando precedente sembra non comparire. Questo è dovuto al fatto che, se non viene specificato esplicitamente, il comando `ifconfig` assegna automaticamente all'interfaccia la maschera di rete corrispondente alla classe a cui appartiene l'indirizzo IP, secondo la suddivisione tradizionale di tab. 7.3, che nel caso sarebbe pari a `255.255.255.0`, dato che l'indirizzo è di classe C.

Quando questa scelta non è corretta si può specificare la rete su cui si affaccia l'interfaccia, indicando esplicitamente la maschera di rete da utilizzare con un comando del tipo:

[18] si tenga presente che con l'uso del nuovo comando `ip address` è possibile assegnare indirizzi multipli alla stessa interfaccia senza dover specificare una interfaccia virtuale nella forma appena vista; in tal caso però non se ne troverà traccia nell'output di `ifconfig`.

```
ifconfig eth0 192.168.1.100 netmask 255.255.0.0
```

Gli esempi precedenti fanno tutti riferimento al caso più comune, quello di una interfaccia per una rete di tipo *broadcast* che viene usata per mettere in comunicazione più macchine su una rete locale. Ma come accennato esistono anche modalità di comunicazione diverse, ed in particolare la sintassi è diversa in caso di collegamenti punto-punto, in cui non esiste una rete locale e due macchine si parlano direttamente, e ciascuna macchina deve essere configurata per dire alla relativa interfaccia quali sono gli IP utilizzati ai due estremi del collegamento.

Direttiva	**Significato**
*[-]*arp	attiva e disattiva (apponendo o no il -) l'uso del protocollo ARP (vedi sez. 7.6.4) per l'interfaccia. Il default è arp.
multicast	abilita il *multicast* sull'interfaccia, normalmente viene selezionato automaticamente quando si attiva l'interfaccia.
*[-]*promisc	attiva e disattiva (apponendo o no il -) la cosiddetta "*modalità promiscua*"[19] per l'interfaccia.
netmask *mask*	imposta la maschera di rete che deve essere specificata in notazione *dotted decimal* come illustrato in sez. 7.2.2.
broadcast *addr*	imposta l'indirizzo di *broadcast* che deve essere specificato in notazione *dotted decimal*.
up	attiva l'interfaccia.
down	disattiva l'interfaccia.
pointopoint *addr*	imposta l'interfaccia in modalità *punto-punto* impostando al contempo l'indirizzo assegnato all'altro estremo.
hw *class addr*	imposta il *MAC address* associato all'interfaccia, il parametro *class* indica il tipo di interfaccia, e normalmente è ether.

Tabella 7.10: Principali direttive del comando ifconfig.

In questo caso non si potrà utilizzare la direttiva netmask, non essendoci una rete locale, mentre occorrerà specificare l'indirizzo dell'altro capo della connessione. Per questo occorre usare la direttiva pointopoint, e se ad esempio si vuole realizzare un collegamento fra due PC con un cavo seriale, si dovrà utilizzare un comando analogo al seguente:

```
ifconfig slip0 192.168.0.1 pointopoint 192.168.0.2
```

ed ovviamente si dovrà usare un comando analogo, con gli indirizzi invertiti, sulla seconda macchina.

In quest'ultimo caso l'uso esplicito di ifconfig è però molto meno diffuso, in quanto la configurazione di una interfaccia *punto-punto* nella maggioranza dei casi è associato all'uso delle le connessioni via modem, e queste normalmente vengono gestite direttamente ad un livello più alto attraverso l'uso di appositi programmi, come vedremo in sez. 7.5.

Gli esempi precedenti ci mostrano anche la sintassi generica dell'invocazione del comando ifconfig. Esso prevede nella forma più comune la specificazione nei primi due argomenti dell'interfaccia di rete e di un indirizzo, mentre negli argomenti successivi possono essere indicati una serie di ulteriori direttive e relativi parametri che consentono di impostare le varie caratteristiche dell'interfaccia. Queste ultime possono essere anche modificate in un secondo tempo,

[19] si chiama così una modalità di funzionamento di una interfaccia di rete su una rete *broadcast* per cui essa riceve tutti i pacchetti che vede passare e non soltanto quelli diretti agli indirizzi ad essa assegnati.

nel qual caso sarà sufficiente utilizzare come primo argomento il nome dell'interfaccia, seguito dalle relative direttive di controllo.

In tab. 7.10 si sono elencate le principali direttive di configurazione disponibili per `ifconfig`, con il formato e la presenza di un eventuale parametro. Si tenga presente che non tutte le direttive possono essere applicabili a qualunque tipo di interfaccia, occorre infatti che la funzionalità sottostante sia disponibile. Per un elenco completo di tutte le direttive, che normalmente vengono lasciate al valore di default, si rimanda comunque alla lettura della pagina di manuale, accessibile al solito con `man ifconfig`.

7.3.2 L'impostazione dell'instradamento ed il comando `route`

Avere attivato una interfaccia di rete ed averle assegnato un indirizzo IP è solo il primo passo; perché sia possibile utilizzare la rete occorre anche impostare *l'instradamento* dei pacchetti. Per questo si usa il comando `route`, che si chiama così proprio perché serve a specificare le strade che essi possono prendere per arrivare a destinazione.

Il comando permette di manipolare la *tabella di instradamento* (o *routing table*) del protocollo IP. Questa è una tabella usata e mantenuta dal kernel per decidere come smistare i pacchetti in uscita e in transito. In sostanza la tabella contiene le associazioni fra le possibili destinazioni di un pacchetto e l'interfaccia di rete che deve essere usata perché questo possa raggiungerle, e per questo le voci che ne fanno parte usualmente vengono chiamate "*rotte*".

In generale le rotte impostate esplicitamente con `route` vengono dette *rotte statiche*, in quanto una volta impostate non vengono più cambiate, in contrapposizione alle *rotte dinamiche* che vengono impostate dai cosiddetti *demoni di routing*,[20] che modificano continuamente le rotte presenti nella tabella di instradamento per tenere conto delle condizioni della rete.

In realtà se è disponibile una sola interfaccia, e non si esce dalla rete locale, non c'è bisogno di eseguire esplicitamente `route`. Infatti tutte le volte che si assegna un indirizzo ad una interfaccia, la rotta per la rete a cui detto indirizzo è associato (detta anche *rotta diretta*) viene creata automaticamente. La cosa vale anche in caso di collegamento *punto-punto*, solo che in tal caso viene inserita una rotta per la singola macchina che si trova all'altro capo della connessione, usando l'IP specificato come parametro per la direttiva `pointopoint` di `ifconfig`. Il problema si pone quando la struttura della rete è più complicata, e non c'è solo la rete locale o collegamenti diretti.

Opzione	Descrizione
`-A`	usa gli indirizzi della famiglia passata come parametro; il default è `inet` per gli indirizzi IP, ma si possono usare gli stessi valori usati da `ifconfig` riportati in tab. 7.9.
`-e`	usa il formato di `netstat` (vedi sez. 7.6.3), ripetuto due volte stampa più informazioni.
`-n`	non effettua la risoluzione degli indirizzi.
`-v`	esegue le operazioni in maniera *prolissa*.

Tabella 7.11: Principali opzioni a riga di comando per l'uso di `route`.

[20] abbiamo accennato in sez. 7.1.3 alla presenza di protocolli usati dai *router* per scambio delle informazioni, i *demoni di routing* sono programmi che implementano questi protocolli e aggiornano automaticamente la tabella di instradamento con le informazioni ottenute.

Il comando `route` prende come primo argomento due direttive che ne specificano le operazioni: `add`, che permette di aggiungere una voce alla tabella, e `del` che ne permette la cancellazione. Se non si specifica nessuna di queste due direttive il comando viene usato soltanto per mostrare lo stato corrente della tabella e non si possono utilizzare nessuna delle ulteriori direttive. Le principali opzioni riconosciute dal comando sono riportate in tab. 7.11.

In generale la direttiva `add` prevede sempre una fra le due opzioni `-net`, per indicare che si intende aggiungere una voce riferita ad una rete, e `-host` per indicare la destinazione verso una singola stazione. A queste deve seguire l'indirizzo, di rete o di nodo, della destinazione, espresso in forma o numerica (usualmente in notazione *dotted decimal*) o simbolica.[21]

Nel caso si sia indicata una rete si deve inoltre specificare sempre anche la relativa *netmask* indicandola (in notazione *dotted decimal*) come parametro per la ulteriore direttiva `netmask`. Se per raggiungere la rete è necessario passare da un *gateway* (nella nostra analogia il centralino per quella destinazione), questo deve essere indicato come argomento per l'ulteriore direttiva `gw`. In genere questo è sempre necessario per impostare il cosiddetto *default gateway*, cioè il *gateway* che si usa per uscire verso tutta Internet. Inoltre si può indicare esplicitamente anche l'interfaccia di rete da utilizzare con la direttiva `dev`, seguita dal nome dell'interfaccia.

Le rotte nella tabella di instradamento sono automaticamente ordinate per dimensione della rete di destinazione, a partire da quella più specifica per arrivare alla più generica ed il kernel invia un pacchetto in uscita sulla prima rotta che comprende nella sua destinazione l'indirizzo IP a cui esso è destinato. Questo vuol dire che la rotta associata al *default gateway*, che per definizione ha come destinazione tutta Internet, essendo la più vasta possibile, quando presente viene sempre utilizzata per ultima.

Le altre principali direttive del comando `route` sono riportate in tab. 7.12; in genere queste sono opzionali e vengono impostate automaticamente agli opportuni valori predefiniti corrispondenti all'interfaccia cui la rotta fa riferimento, oppure sono appannaggio dei demoni di *routing* ed inutilizzate per le rotte statiche. Fa eccezione la direttiva `reject` che viene utilizzata per impostare una cosiddetta *blocking route*, cioè una rotta che fa sì che il tentativo di inviare i dati su quella rete fallisca con un errore di instradamento. In genere la si usa per bloccare l'accesso dei pacchetti a certe reti prima di arrivare alla *default route*.

Direttiva	**Descrizione**
`netmask` *n.n.n.n*	imposta la maschera di rete.
`gw` *n.n.n.n*	imposta l'indirizzo di un *gateway*.
`reject`	installa una rotta bloccata.
`metric` *N*	imposta il valore del campo `Metric`.
`mss` *N*	imposta la *Maximum Segment Size* per i pacchetti TCP che usano la rotta (in byte).
`window` *N*	imposta la dimensione della *advertizing window* del TCP per la rotta (in byte).
`irtt` *T*	imposta il *Round Trip Time* iniziale per la rotta (in millisecondi).
`mod,dyn,reinstate`	flag diagnostici usati dai demoni di *routing*.
`dev` *iface*	imposta l'interfaccia usata per raggiungere la destinazione.

Tabella 7.12: Direttive del comando `route`.

[21] quest'ultima però richiede che sia disponibile una *risoluzione* dello stesso (tratteremo l'argomento in sez. 7.4) cosa che non sempre è possibile, pertanto è preferibile usare sempre la forma numerica.

Le opzioni di base `-net` e `-host` valgono anche per la direttiva di cancellazione `del`, la differenza è che mentre con `add` devono essere specificate interamente le caratteristiche della rotta che si vuole aggiungere, per cancellarne una già presente basta identificare univocamente la voce relativa perché il comando abbia effetto.

Il concetto fondamentale del *routing* è che ciascun nodo sulla rete quando deve inviare un pacchetto deve sapere se questo può raggiungere direttamente la destinazione, come avviene nel caso di una rotta diretta relativa alla propria rete locale o a macchine per cui si ha un collegamento fisico, o se invece per inviare il pacchetto verso quella destinazione è necessario rivolgersi ad un nodo limitrofo (il *gateway*) che sia in grado di mandarlo a destinazione.

In sostanza per ciascun pacchetto che una macchina deve smistare deve essere disponibile la "segnaletica stradale" che indichi da quale parte svoltare per arrivare a destinazione. In realtà quando si ha a che fare con una rete locale le cose sono molto più semplici, e, tornando all'analogia telefonica, tutto quello che serve è sapere qual'è il numero del centralino per uscire (il *gateway*) e quali sono i numeri interni.

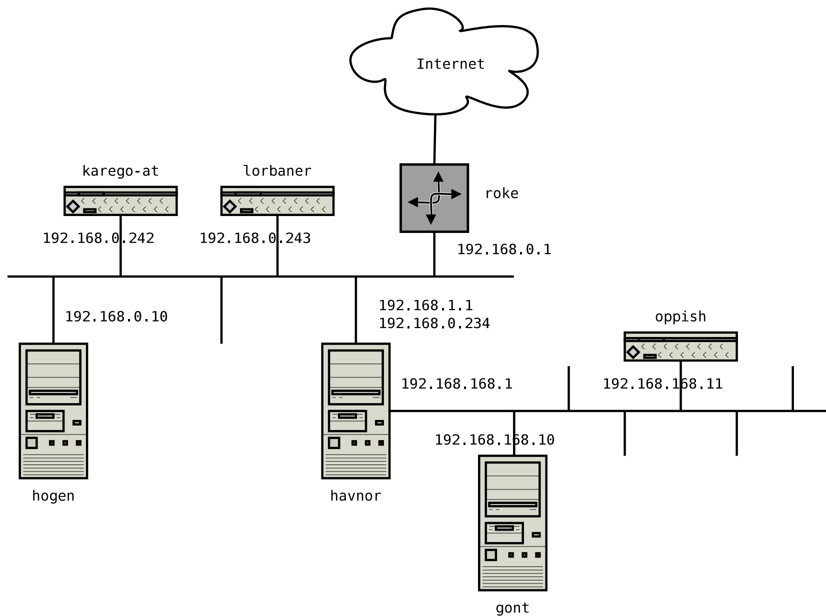


Figura 7.4: Schema di una rete di prova.

Per chiarire meglio questi concetti supponiamo di avere la rete schematizzata in fig. 7.4, che prenderemo come riferimento per i prossimi esempi, e vediamo come deve essere configurato l'instradamento per le varie macchine di cui la rete è composta. In questo caso si vede che un ruolo particolare è rivestito dalla macchina `havnor`, che ha due interfacce ed è posta a cavallo fra due reti diverse. Nel nostro esempio essa dovrà fare da ponte fra le due reti,[22] passando i

[22]tecnicamente si parla di un *bridge* quando si ha a che fare con un apparato che fa da *ponte* fra due sotto-reti fisiche passando i pacchetti dall'una all'altra a livello di *datalink* (in genere sui protocolli della famiglia IEEE 802.x), in maniera completamente trasparente ai livelli superiori; Linux può essere utilizzato anche in questo

pacchetti da una interfaccia ad un'altra, e svolgerà quindi un ruolo di *gateway*.

Perché una macchina possa passare pacchetti di rete da una interfaccia ad un'altra deve però essere attivato nel kernel il cosiddetto *IP forwarding*, che di default è disabilitato. Questo si può fare attraverso l'interfaccia del *sysctl* (trattata in sez. 3.1.3), o usando direttamente i file con cui questa viene rappresentata nel filesystem `/proc`. Nel caso specifico per abilitare l'*IP forwarding* basterà eseguire il comando:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Una volta fatto questo potremo passare a configurare le rotte della rete in fig. 7.4 usando `route`. Il primo passo è sempre quello di controllare la situazione corrente. Quando è invocato senza parametri il comando mostra il contenuto corrente della *tabella di instradamento*. Ad esempio, nel caso illustrato in fig. 7.4, andando su `havnor` avremo:

```
root@havnor:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.168.0   0.0.0.0         255.255.255.0   U     0      0        0 eth1
0.0.0.0         192.168.0.1     0.0.0.0         UG    0      0        0 eth0
```

dove si è usata l'opzione `-n` per avere una stampa con i valori numerici per gli indirizzi, con l'uscita del comando che mostra le voci presenti nella tabella di instradamento.

Nome	**Descrizione**
`Destination`	rete o nodo di destinazione.
`Gateway`	indirizzo del *gateway* (`0.0.0.0` se non impostato).
`Genmask`	*netmask* della rete di destinazione
`Flags`	flag associati alla rotta (vedi tab. 7.14).
`Metric`	metrica della rotta (distanza dalla destinazione in numero di salti).
`Ref`	numero di riferimenti nella tabella di instradamento (non usato nel kernel).
`Use`	numero di verifiche sulla rotta.
`Iface`	interfaccia verso cui sono inviati i pacchetti.
`MSS`	valore di default per l'MSS (la *Maximum Segment Size*) delle connessioni TCP su questa rotta.
`Window`	default per la *window size* delle connessioni TCP su questa rotta.
`irtt`	valore iniziale dell'RTT (*Round Trip Time*) per il protocollo TCP.

Tabella 7.13: Nomi della colonna e relativo significato per le varie informazioni riportate nell'uscita del comando `route`.

Se non specificato altrimenti il comando tenta anche di risolvere (vedi sez. 7.4) i nomi delle macchine e delle reti. L'output usa la formattazione appena mostrata in cui la prima colonna identifica la destinazione, la seconda il *gateway*, la terza la sottorete coperta dalla destinazione, la quarta lo stato della rotta e l'ultima l'interfaccia usata per l'invio dei pacchetti. Il significato

---

modo, abilitando l'opportuno supporto nel kernel, qui però stiamo parlando semplicemente del passaggio di un pacchetto da una interfaccia ad un'altra a livello di rete, che invece è il lavoro svolto dai *router*.

delle altre colonne è associato agli aspetti più sofisticati del *routing*, quelli gestiti dai *routing daemon*. Uno specchietto delle principali informazioni mostrate dal comando è in tab. 7.13, maggiori dettagli sono nella pagina di manuale.

Nell'esempio mostrato in precedenza si può notare come ci siano tre diverse destinazioni associate a tre diverse sotto-reti, due delle quali fanno capo alla stessa interfaccia (quella che in sez. 7.3.1 abbiamo visto essere *multihomed*). Per tutte queste sotto-reti, che sono accessibili direttamente da una interfaccia locale, non esiste un *gateway* e questo è indicato dall'uso dell'indirizzo generico in seconda colonna.

L'ultima riga indica la *rotta di default* e presenta il *default gateway* con l'indicazione dell'indirizzo della macchina (che come accennato deve essere raggiungibile con le rotte precedenti) a cui devono essere inviati i pacchetti che devono uscire su Internet. Come abbiamo già accennato in sez. 7.2.2 l'indirizzo di rete nullo con *netmask* nulla corrisponde ad una rete in cui tutti e 32 i bit disponibili sono utilizzati come indirizzi delle macchine al suo interno, e quindi questa rotta indica appunto tutta Internet.

Simbolo	**Significato**
`U`	la rotta è attiva.
`H`	la destinazione è una macchina singola.
`G`	usa un *gateway*.
`R`	rotta reintegrata da un instradamento dinamico.
`D`	installata dinamicamente da un demone o da una redirezione.
`M`	modificata da un demone o una redirezione.
`A`	installata da `addrconf`.
`C`	voce nella cache.
`!`	rotta bloccata (impedisce l'instradamento per la destinazione specificata).

Tabella 7.14: Significato dei simboli utilizzati nella colonna `Status` del comando `route`.

Per tutti gli altri computer nella rete, che hanno una sola interfaccia di rete, una volta assegnato l'indirizzo con `ifconfig`, tutto quello che resta da fare è solo specificare qual'è l'indirizzo del *default gateway* che si usa per uscire su Internet (di solito quello del *router* fornito dal provider). Nel caso della rete di esempio illustrata in fig. 7.4 questo si può fare con il comando:

```
route add default gw 192.168.0.1
```

che ovviamente deve essere eseguito dall'amministratore, in quanto cambiare i contenuti della tabella di instradamento è una operazione privilegiata.

Il comando dovrà essere ripetuto per tutte le macchine mostrate nel tratto di rete di sinistra in fig. 7.4, ma vedremo in sez. 8.2 come queste impostazioni possono essere automatizzate usando il servizio DHCP. L'effetto di questo comando è quello di creare su ciascuna di esse una *rotta di default*, identica a quella riportata nell'ultima riga dell'esempio di tabella di instradamento di pag. 511, perché `default` non è altro che l'abbreviazione di "`-net 0.0.0.0 netmask 0.0.0.0`".

Le macchine sulla rete `192.168.168.0` però hanno il problema di non poter vedere direttamente il *router*, dato che questo è posto sulla rete `192.168.0.0`. Inoltre esse non possono neanche essere raggiunte delle macchine di quella rete, che non hanno un accesso diretto alla rete `192.168.168.0`. In questo caso infatti i pacchetti, per poter passare da una rete all'altra, devono attraversare `havnor`, che, con le sue due interfacce di rete, fa da *router* fra i due tratti.

Come in tutti i casi in cui l'accesso ad una certa rete è condizionato al passaggio da una macchina specifica che fa da *router*, si deve allora specificare una rotta statica che indichi anche quest'ultima come *gateway* per quella rete. Così se da `lorbaner` si vuole accedere a `gont`, vi si dovrà impostare una rotta statica con:

```
root@lorbaner:~# route add -net 192.168.168.0 netmask 255.255.255.0 gw 192.168.0.234
```

che dice che per i pacchetti destinati alla rete `192.168.168.0` si deve usare come *gateway* la macchina `192.168.0.234`. In questo modo, essendo la rotta per la rete `192.168.168.0` più specifica di quella di default questi, invece che a `roke`, verrebbero inviati ad `havnor`, che può reinviarli a destinazione avendo una interfaccia su quella rete.

Questo però non è detto sia sufficiente, perché consente solo ai pacchetti che partono da `lorbaner` di raggiungere `gont`, ma per poter realizzare un collegamento effettivo fra le due macchine devono anche poter tornare indietro le risposte. Si deve allora anche dire a `gont` come fare a raggiungere la rete `192.168.0.0` sui cui lui non ha accesso diretto. Per far questo si potrebbe pensare di impostare anche in questo caso una rotta statica per questa rete con un comando analogo al precedente come:

```
root@gont:~# route add -net 192.168.0.0 netmask 255.255.255.0 gw 192.168.168.1
```

Questo sarebbe certamente sufficiente ad ottenere il risultato di garantire la comunicazione fra `lorbaner` e `gont`, ma resta aperto il problema di come impostare il *default gateway* di `gont`. Se infatti tentasse di usare direttamente l'indirizzo di `roke` (`192.168.0.1`) non essendo questo raggiungibile direttamente, si avrebbe un errore, anche se adesso è presente una rotta per quella rete.

Si può però utilizzare di nuovo `havnor` che ha, come risulta nella tabella di instradamento illustrata nell'esempio di pag. 511, anche una rotta per raggiungere Internet, e che è raggiungibile direttamente, impostando la rotta di default con il comando:

```
root@gont:~# route add default gw 192.168.168.1
```

e questo risolverebbe il problema in quanto stavolta `havnor` riceverebbe i pacchetti indirizzati verso Internet e li invierebbe lui a `roke`.[23]

Ma a questo punto il precedente comando risulta inutile, in quanto la rete `192.168.0.0` è senz'altro compresa nella rotta di default, ed il *gateway* è lo stesso. Pertanto basterebbe quest'ultimo comando per assicurare il collegamento sia verso le macchine della rete `192.168.0.0` che verso Internet. La differenza starebbe nel fatto che nel primo caso `havnor` potrebbe inviare il pacchetto direttamente a destinazione mentre nel secondo dovrebbe passare da `roke`, ma questo, per quanto concerne la configurazione di `gont`, è del tutto irrilevante.

7.3.3 La configurazione della rete all'avvio del sistema.

Nelle sezioni precedenti abbiamo esaminato i comandi che consentono di eseguire manualmente la configurazione della rete su una macchina Linux, nelle operazioni ordinarie essi però non vengono usati direttamente quasi mai, in quanto la configurazione viene effettuata quasi sempre in maniera automatica. In realtà i casi più comuni che si presentano sono sostanzialmente due:

[23] di nuovo perché poi le risposte possano tornare indietro occorre anche che `roke` possa raggiungere la rete `192.168.168.0`, cosa che si fa inserendo anche su di esso una rotta statica come fatto per `lorbaner`.

- Un collegamento diretto ad Internet con una connessione attraverso un modem analogico, ISDN o ADSL connesso alla macchina stessa.[24] In questo caso il collegamento può anche non essere permanente, nel qual caso lo si potrà o dovrà attivare a richiesta con un opportuno programma.

- Un collegamento all'interno di una rete locale, usualmente effettuato tramite una scheda di rete Ethernet. In questo caso il collegamento, almeno quello alla rete locale, è permanente, e viene generalmente attivato dagli script di avvio.

In entrambi i casi sono comunque richiesti alcuni dati necessari a configurare la rete. Nel primo caso questi devono essere stati forniti dal provider; ad esempio nel caso di un modem analogico serve il numero telefonico da contattare, ed in genere è anche prevista una qualche modalità di autenticazione che richiede username e password del proprio account. A questi dati si possono eventualmente aggiungere quelli relativi alla inizializzazione del modem utilizzato, anche se ormai tutte le distribuzioni sono in grado di eseguire un riconoscimento automatico nella maggior parte dei casi.

In questo caso non è necessario preoccuparsi di conoscere in anticipo l'indirizzo IP assegnato alla propria macchina in quanto sarà il programma di connessione, in genere utilizzando il comando `pppd` che tratteremo in sez. 7.5.2, ad utilizzare le informazioni che gli vengono fornite dal provider per eseguire la configurazione della rete. Al più ci potrà essere da configurare a mano, per quei pochi provider che non forniscono l'informazione, gli indirizzi dei DNS (torneremo su questo in sez. 7.4.2).

Nel secondo caso invece sono necessarie le informazioni relative alla rete all'interno di cui ci si trova, che potrebbe anche essere semplicemente quella fornita dall'interfaccia Ethernet del proprio *router* ADSL, ed in particolare l'indirizzo IP da assegnare alla macchina, la *netmask*, e l'indirizzo del *gateway*. In realtà se sulla rete è disponibile un server DHCP (torneremo sull'argomento in sez. 8.2) l'indirizzo non dovrà essere specificato a mano, ma potrà essere impostato automaticamente grazie ad esso. Tutti i programmi di configurazione prevedono questa possibilità, nel qual caso non è necessario fornire nessuna informazione specifica, se non quella di usare il DHCP.

Si tenga presente inoltre che nel caso si debba creare una propria rete locale, di cui poi si sarà gli amministratori, si dovrà anche effettuare la scelta degli indirizzi che si vogliono utilizzare al suo interno prima di assegnarli alle varie macchine. In questo caso è sempre opportuno usare una delle reti appartenenti alle classi riservate elencate in tab. 7.6, che la IANA ha destinato appositamente a questo scopo, e che non vengono mai assegnati a macchine pubbliche su Internet.

Questa è una regola da seguire sempre anche per una rete di cui non è previsto alcun accesso ad Internet, per la quale in teoria si potrebbe pensare di usare indirizzi qualunque. Usare degli indirizzi pubblici per una rete privata è comunque sbagliato perché la volta che si dovesse fare accedere ad Internet una delle macchine di questa rete, gli indirizzi pubblici presenti su Internet corrispondenti alla propria rete privata non sarebbero mai raggiungibili.

A parte il caso di connessioni non permanenti da attivare a richiesta, in generale la rete viene configurata nelle fasi iniziali della procedura di avvio del sistema tramite gli opportuni script

[24] si tenga presente che si sta parlando di *modem*, anche nel caso di ADSL, nel caso si abbia invece un *router* ADSL, come avviene ormai sempre più spesso, si ricade nel caso successivo.

Distribuzione	Script di avvio	Altre distribuzioni
Debian	`/etc/init.d/networking`	Ubuntu, Mint
RedHat	`/etc/init.d/network`	SuSE, Scientific Linux

Tabella 7.15: Gli script di inizializzazione della rete per varie distribuzioni.

(vedi sez. 5.3.5) in modo da garantirne l'accesso al successivo avvio dei servizi che la usano.[25] Ogni distribuzione utilizza una sua versione di questi script ed in tab. 7.15 si sono riportati quelli usati dalle distribuzioni più diffuse.

A loro volta gli script leggono da opportuni file di configurazione tutte le informazioni necessarie, come le interfacce da configurare, gli IP ad esse assegnati e le netmask delle relative reti, l'indirizzo del *default gateway*, la presenza di eventuali ulteriori rotte statiche, ecc. Pertanto per modificare in maniera permanente le impostazioni della rete si devono modificare i file in cui sono memorizzate le informazioni usate dagli script di avvio. Eventuali modifiche provvisorie si possono sempre effettuare con i singoli comandi illustrati nelle sezioni precedenti, ma ovviamente al successivo riavvio tutti i cambiamenti saranno perduti.

Anche i file di configurazione della rete variano da distribuzione a distribuzione, così come può essere diverso il loro formato. Prenderemo in esame due dei casi più comuni, Debian e RedHat le cui scelte, così come per gli script di avvio, sono riprese anche da altre distribuzioni, sempre secondo lo schema di tab. 7.15. In genere i programmi di configurazione automatica (o i vari programmi grafici per la configurazione) non fanno altro che leggere e modificare i valori che stanno su questi file con una qualche interfaccia semplificata. Dato che si tratta come sempre di file di testo, è comunque sufficiente saper usare un editor qualunque.

Debian Il file di configurazione delle interfacce è `/etc/network/interfaces`, il cui formato è descritto in dettaglio dalla omonima pagina di manuale accessibile con `man interfaces`. Il file consente di controllare con grande flessibilità la configurazione delle interfacce, ed è in genere suddiviso in sezioni introdotte da opportune parole chiave, le principali delle quali sono riportate in tab. 7.16. Ciascuna sezione può richiedere eventuali ulteriori direttive di configurazione che in genere si specificano scrivendole con una indentazione sulle righe successive alla parola chiave che definisce la relativa sezione.

Direttiva	Significato
`auto`	dichiara una lista di interfacce che deve essere avviata automaticamente all'avvio (dal comando `ifup -a`).
`allow-hotplug`	dichiara una lista di interfacce che devono essere configurate automaticamente tramite *hotplug* (vedi sez. 5.4.5).
`mapping`	definisce le modalità per creare una corrispondenza fra un nome logico assegnato ad una interfaccia di rete ed una specifica interfaccia fisica.
`iface`	dichiara la configurazione di una interfaccia.

Tabella 7.16: Principali direttive che identificano le diverse sezioni di `/etc/network/interfaces`.

[25] questo non è sempre vero nel caso di sistemi desktop in cui in genere la configurazione viene eseguita al rilevamento di una possibile connessione da opportuni programmi di gestione integrati nell'ambiente desktop stesso; non tratteremo qui questa casistica che dipende dal programma utilizzato per eseguire questo compito.

Un esempio del contenuto di questo file, nel caso di una macchina con una sola interfaccia di rete a cui si vuole assegnare un indirizzo fisso, configurandone al contempo i parametri per l'accesso ad Internet, potrebbe essere il seguente:

/etc/network/interfaces

```
auto lo eth0
iface lo inet loopback
iface eth0 inet static
    address 172.16.1.12
    netmask 255.255.255.0
    gateway 172.16.1.1
```

La prima riga, introdotta dalla parola chiave `auto`, dice quali sono le interfacce da attivare automaticamente all'avvio del sistema, che possono essere specificate sia insieme, come nell'esempio, che in altrettante righe distinte. Le due righe seguenti, introdotte dalla parola chiave `iface`, servono a impostare i parametri di ciascuna interfaccia. Per ciascuna interfaccia deve essere fornita una riga in cui specificarne il nome, ed a seguire la famiglia di protocolli usata e le modalità di configurazione da indicare con opportune parole chiave separate da spazi.

Nel caso dell'esempio la famiglia di protocolli utilizzata per entrambe le interfacce (`lo` e `eth0`) è sempre `inet`, che indica l'usuale TCP/IP con IPv4, ma sarebbe stato possibile usare anche `ipx` per indirizzi IPX, o `inet6` per IPv6.

La configurazione delle due interfacce differisce invece nelle modalità con cui questa viene effettuata, ed a seconda di quale si è scelta potrà essere necessario indicare ulteriori parametri di configurazione; le principali modalità disponibili sono sostanzialmente le seguenti:

`loopback` configurazione per il *localhost*, si usa soltanto per l'interfaccia di loopback e non è necessario indicare altro.

`dhcp` si richiedono i dati di configurazione ad un server DHCP con una opportuna richiesta tramite questo protocollo (torneremo sull'argomento in sez. 8.2).

`manual` l'interfaccia non deve essere configurata di default, la si usa ad esempio per interfacce che fanno parte di un *bridge*.

`static` si assegnano i valori in maniera statica secondo i parametri specificati nelle righe seguenti.

per le prime due modalità non occorre specificare altro,[26], mentre se si sceglie una assegnazione statica dell'indirizzo IP con `static` si dovranno impostare gli opportuni parametri nelle righe seguenti, che di solito si indentano per maggiore chiarezza, come nell'esempio.

Nel caso in esempio si sono indicati l'indirizzo, la maschera di rete e l'indirizzo del *default gateway* utilizzando rispettivamente le parole chiave `address`, `netmask` e

[26] in realtà con `dhcp` si possono passare delle richieste specifiche al server (vedi sez. 8.2.2) utilizzando alcuni parametri, ma in genere questi non vengono mai utilizzati, essendo sufficienti i valori di default.

`gateway`. Quest'ultimo deve essere specificato una volta sola, per l'interfaccia da cui è raggiungibile. Si sono riassunte in tab. 7.17 le principali direttive disponibili.

Direttiva	**Significato**
`address`	indirizzo dell'interfaccia.
`netmask`	*netmask* della rete.
`network`	indirizzo di rete (opzionale).
`gateway`	*default gateway*.
`broadcast`	indirizzo di *broadcast* (opzionale).
`hwaddress`	*MAC address* da assegnare alla scheda.

Tabella 7.17: Principali direttive per la configurazione in modalità `static`.

Per ciascuna interfaccia, indipendentemente dal tipo di indirizzi o dalla modalità di configurazione, possono inoltre essere specificati dei comandi da chiamare contestualmente all'attivazione e alla disattivazione dell'interfaccia, questo si fa in maniera analoga alla precedente indicazione di parametri, usando le parole chiave `up`, `pre-up`, `down`, `post-down` seguite dal comando, che verrà eseguito rispettivamente dopo e prima dell'attivazione e prima e dopo la disattivazione.

Si tenga presente poi che le interfacce associate alle connessioni punto-punto effettuate tramite modem non vengono normalmente menzionate in questo file, trattandosi in genere di connessioni non permanenti attivate a richiesta. Con la diffusione dei modem ADSL però nelle versioni più recenti è possibile attivare una connessione punto-punto permanente usando la nuova modalità `ppp`. In questo caso viene comunque utilizzata una configurazione esistente di `pppd` (tratteremo l'argomento in sez. 7.5.2), ed occorrerà specificare sulla riga seguente la direttiva `provider` con argomento il nome del file di direttive sotto `/etc/ppp/peers` che deve essere usato.

RedHat i file per la configurazione all'avvio delle interfacce di rete sono mantenuti nella directory `/etc/sysconfig/network-scripts`, e ce n'è uno per interfaccia con un nome del tipo `ifcfg-`*`iface`*, dove *`iface`* è il nome dell'interfaccia da configurare. Ciascuno di questi contiene poi i relativi parametri, indicati sempre nella forma di assegnazione di un valore ad una variabile di shell,[27] il cui significato è espresso in maniera evidente dal relativo nome per gran parte di esse.

Il contenuto del file dipende in genere dal tipo di interfaccia, a seconda dei parametri necessari alla configurazione delle interfacce di quel tipo. Il caso più comune è quello delle interfacce di tipo Ethernet (ma i parametri sono identici per altre tecnologie simili); in questo caso una variabile sempre presente deve essere `DEVICE` che definisce il nome dell'interfaccia e deve corrispondere a quello del nome del file. Una seconda variabile fondamentale è `BOOTPROTO` che indica se deve essere usato un protocollo di auto-configurazione, coi valori `bootp` o `dhcp` dai nomi dei rispettivi protocolli, (vedi sez. 8.2.1) oppure una configurazione statica, con il valore `none`. Un'altra variabile importante è `ONBOOT` che segnala, con il valore `yes`, che l'interfaccia deve essere configurata automaticamente all'avvio.

[27] in effetti si tratta proprio di questo, il file viene letto dagli script di avvio che usano queste variabili di shell per effettuare la configurazione.

Pertanto nel caso si debba usare l'autoconfigurazione all'avvio con il protocollo DHCP un esempio del contenuto del file `ifcfg-eth0`, relativo alla prima interfaccia Ethernet, potrebbe essere il seguente:

```
ifcfg-eth0
DEVICE=eth0
BOOTPROTO=dhcp
ONBOOT=yes
```

Se viceversa si volesse effettuare una configurazione statica si dovrebbero specificare in altrettante variabili l'indirizzo IP, la *netmask*, e l'indirizzo di *broadcast* (rispettivamente `IPADRR`, `NETMASK` e `BROADCAST`), inoltre si dovrebbe indicare un eventuale *default gateway* con la variabile `GATEWAY`; in tal caso un possibile esempio del contenuto del file `ifcfg-eth0` potrebbe essere il seguente:

```
ifcfg-eth0
DEVICE=eth0
BOOTPROTO=none
ONBOOT=yes
IPADDR=172.16.1.12
NETMASK=255.255.255.0
GATEWAY=172.16.1.1
BROADCAST=172.16.1.255
USERCTL=no
```

dove è definita anche la variabile `USERCTL` che serve a marcare (se assegnato al valore `yes`) le interfacce che anche agli utenti normali possono attivare e disattivare con il comando `ifup`. Un elenco delle principali variabili di controllo è riportato in tab. 7.18.

Variabile	**Significato**
`BOOTPROTO`	protocollo di auto-configurazione (`bootp`, `DHCP`).
`BROADCAST`	indirizzo di *broadcast*.
`DEVICE`	nome dell'interfaccia.
`GATEWAY`	indirizzo IP del *default gateway*.
`HWADDR`	indica il *MAC address* della scheda (da usare su macchine con più schede per essere sicuri dell'assegnazione all'interfaccia corretta).
`IPADDR`	indirizzo IP da assegnare alla macchina.
`NETMASK`	*netmask* per la rete locale.
`NETWORK`	indirizzo della rete (opzionale).
`ONBOOT`	attivazione (`yes`) o meno (`no`) all'avvio.
`USERCTL`	attivazione (`yes`) o meno (`no`) da parte di un utente non privilegiato.

Tabella 7.18: Principali variabili di controllo per i file `ifcfg-ethN`.

Anche se gli script di avvio ed i file di configurazione variano da distribuzione a distribuzione sono in genere disponibili due comandi standardizzati,[28] `ifup` e `ifdown`, che consentono rispet-

[28] in realtà solo su Debian si tratta di un vero e proprio comando, mentre su RedHat e SuSE si tratta di un semplice script di shell.

tivamente di attivare o disattivare un'interfaccia, eseguendo al contempo la sua configurazione secondo quanto previsto nei file di configurazione della rete appena trattati. Nella loro forma generica questi comandi prendono come unico argomento il nome dell'interfaccia da attivare o disattivare, le varie distribuzioni poi hanno realizzato versioni di questi comandi che possono prevedere altri parametri o opzioni, per i quali si consiglia di consultare la relativa pagina di manuale.

7.3.4 Comandi di configurazione avanzata

Finora abbiamo trattato le configurazioni più elementari della rete, in cui i tradizionali comandi `ifconfig` e `route` sono più che sufficienti. Anche se il loro uso continua a persistere, visto che comunque vanno bene nella gran parte dei casi, questi comandi sono stati indicati come deprecati per molti anni dato che non sono in grado di gestire configurazioni di rete più complesse: ad esempio avendo due router indipendenti per arrivare su internet non consentono di definire una configurazione che consenta di utilizzarli entrambi.

Come loro sostituto è stato introdotto il comando `ip`, un comando generico in grado di consentire la gestione di tutte le funzionalità di rete più avanzate. Data la sua ampiezza questo comando richiede che si usi il primo argomento come una sorta di sottocomando per indicare su quale "*oggetto*" operare; ad esempio si potranno configurare gli indirizzi di rete con `ip address` e le rotte con `ip route`. Si sono riassunti gli "*oggetti*" (o sottocomandi) più rilevanti in tab. 7.19.

Il comando supporta anche alcune opzioni, che devono essere specificate prima di indicare un oggetto, la più rilevante (per l'elenco completo si consulti `man ip`) è `-f` seguita dal nome del protocollo su cui si vuole operare, secondo i valori di tab. 7.9. Il comando infatti cerca di desumere il protocollo dagli argomenti che gli vengono passati, e qualora questo non sia possibile usa IPv4 come default, ma ci sono casi in cui occorre usare l'opzione per dare una indicazione esplicita, o si vuole ridurre le informazioni stampate ai soli dati del protocollo specificato.

Oggetto	Significato
`address`	manipola gli indirizzi di rete associati alle schede.
`help`	consente di ottenere la documentazione di un altro sottocomando.
`neighbour`	manipola la tabella di ARP e sostituisce il comando `arp` (vedi sez. 7.6.4).
`link`	manipola le caratteristiche di una interfaccia al livello di collegamento fisico.
`route`	manipola le tabelle di *routing* e sostituisce il comando `route`.
`rule`	manipola le regole di *routing*, che consentono di realizzare politiche di *routing* complesse.

Tabella 7.19: Identificativi dei principali oggetti del comando `ip`.

La sintassi prevede che non sia necessario scrivere per intero il primo argomento ma, fintanto che non c'è ambiguità, solo le prime lettere. Come secondo argomento deve essere specificata una azione da compiere, che dipende dall'oggetto specificato nel primo. Oltre a quelle specifiche sono previste alcune azioni generiche, presenti per la gran parte degli oggetti, come `add` per aggiungere, `delete` per rimuovere e `show` (omonima di `list`) per fare un elenco; si sono riassunte le principali in tab. 7.20. Se non si indica nulla viene utilizzata come default `show`. Negli argomenti successivi devono essere indicate eventuali ulteriori direttive di configurazione, espresse nella forma di elenco di coppie "`direttiva valore`" separate da spazi; l'ordine in cui le si esprimono non ha rilevanza.

Oggetto	Significato
`add`	aggiunge un oggetto (indirizzo, rotta, regola, ecc).
`del`	rimuove un oggetto (indirizzo, rotta, regola, ecc).
`flush`	rimuove tutti i contenuti (indirizzi, rotte, regole, ecc).
`show`	elenca i contenuti (indirizzi, rotte, regole, ecc).

Tabella 7.20: Principali azioni generiche disponibili per i vari sottocomandi di `ip`.

La configurazione degli indirizzi di rete si effettua con `ip address`, che in sostanza è il sostituto di `ifconfig`. Se invocato senza ulteriori argomenti il comando mostra l'elenco di tutte le interfacce e dei relativi indirizzi. In tutti gli altri casi occorre indicare su quale interfaccia operare con l'ulteriore direttiva `dev`; questa indicazione è obbligatoria anche per visualizzare le caratteristiche di una singola interfaccia, ad esempio:

```
root@hogen:~# ip addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 74:2b:62:7b:62:db brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.10/24 brd 192.168.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::762b:62ff:fe7b:62db/64 scope link
       valid_lft forever preferred_lft forever
```

elenca sia il *MAC address* della scheda che i due indirizzi assegnati sull'interfaccia `eth0` della macchina `hogen`; quello IPv4, classificato come `scope global` per indicarne la validità come indirizzo generico, e quello IPv6 classificato come `scope link` in quanto è quello *link-local* associato direttamente alla scheda di rete.

Oggetto	Significato
`broadcast`	indica esplicitamente l'indirizzo di *broadcast*, si possono usare i valori "+" e "-" per ottenerlo portando rispettivamente a 1 e a 0 tutti i bit della parte locale (i meno significativi) dell'indirizzo, se non si indica si sottintende "+".
`dev`	indica l'interfaccia su cui operare.
`label`	indica una etichetta da associare all'indirizzo, consente di mantenere la compatibilità con `ifconfig` per gli *IP alias* usando stringhe nella forma *`ifname`*`:`*`N`*.
`local`	il default, indica l'indirizzo (locale) da assegnare all'interfaccia; normalmente viene omesso.
`peer`	indica l'indirizzo dell'altro capo di una connessione punto-punto.

Tabella 7.21: Principali direttive per `ip address`.

Per assegnare un indirizzo di rete alla scheda occorre specificare l'azione `add` seguita dall'indirizzo, che può essere sia IPv4 che IPv6 (andrebbe indicato con la direttiva `local` ma questo è il default e si può omettere). Inoltre l'interfaccia deve essere sempre indicata con la direttiva `dev`. Nel caso di configurazione di connessione punto-punto occorrerà impostare l'indirizzo dell'altro capo della connessione con la direttiva `peer` *`ADDRESS`*, si sono riassunte le principali in tab. 7.21. A differenza di `ifconfig` non è necessaria una direttiva separata per indicare la *netmask*, basterà indicarla esprimendo l'indirizzo in notazione CIDR, seguito da una "`/`" e dal numero di bit, un esempio del comando è il seguente:

```
ip addr add 192.168.0.11/24 dev eth0
```

La differenza più importante è però nella gestione dell'*IP aliasing*: non esistono più infatti nomi "*speciali*" per l'interfaccia, come `eth0:1`, da usare per gli indirizzi aggiuntivi e basta semplicemente ripetere l'assegnazione indicando un altro indirizzo, che verrà aggiunto; una volta eseguito il comando precedente su `hogen`, il risultato sarà:

```
root@hogen:~# ip -f inet addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    inet 192.168.0.10/24 brd 192.168.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet 192.168.0.11/24 scope global secondary eth0
       valid_lft forever preferred_lft forever
```

Si tenga presente che tutti gli indirizzi aggiunti ad una scheda che sono nella stessa rete di uno già presente ne diventano dei secondari, marcati, come nell'esempio precedente, "`secondary`". La conseguenza è che vengono rimossi automaticamente quando viene rimosso il loro primario. Inoltre gli indirizzi così assegnati non vengono riconosciuti da `ifconfig`, a meno che non vi si aggiunga una etichetta con la ulteriore direttiva `label` indicando un nome nella forma tradizionale. Pertanto se fosse voluto mantenere la visibilità con `ifconfig` si sarebbe dovuto aggiungere l'indirizzo con il comando:

```
ip addr add 192.168.0.11/24 dev eth0 label eth0:1
```

La cancellazione degli indirizzi si effettua invece usando l'azione `del`; se non si specifica altro (ma l'interfaccia deve comunque essere indicata con `dev`) verrà rimosso il primo indirizzo, altrimenti lo si dovrà indicare sul resto della riga di comando, usando la stessa sintassi di `add`. Si possono anche cancellare tutti gli indirizzi assegnati in un colpo solo con l'azione `flush`, selezionando eventualmente una singola interfaccia su cui operare con `dev`.

Si noti però come `ip addr` non copra tutte le funzionalità di `ifconfig` che, oltre ad impostare gli indirizzi di una interfaccia, consente anche di configurarne le caratteristiche relative al livello di collegamento fisico, come modificare la MTU, o attivarla o disattivarla, bloccare l'uso del protocollo ARP (vedi sez. 7.6.4). Queste funzionalità sono state scorporate dalla gestione degli indirizzi e vengono coperte dal comando `ip link`. Al solito, se eseguito senza ulteriori argomenti, verrà eseguita l'azione di default `show`, che mostra lo stato delle interfacce, riportando solo i dati relativi al livello fisico che vengono mostrati anche con `ip addr`:

```
root@hogen:~# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 74:2b:62:7b:62:db brd ff:ff:ff:ff:ff:ff
```

Benché si possano utilizzare le azioni `add` e `del` per creare vari tipi di interfacce virtuali ed assegnar loro opportune proprietà, l'uso più comune del comando è con l'azione `set` che consente di impostare le caratteristiche di una interfaccia fisica, che al solito deve essere indicata con la direttiva `dev`. In particolare le due direttive `up` e `down` servono per mandare rispettivamente su e giù una interfaccia, come le analoghe di `ifconfig`. Si può poi usare la direttiva `mtu` per impostarne la *Maximum Transfer Unit* (vedi sez. 7.6.2), o assegnargli un nome diverso con la direttiva `name`, si sono riassunte le principali direttive supportate da `set` in tab. 7.22.

La gestione delle rotte di instradamento viene invece effettuata con `ip route`, che ricomprende le funzionalità di `route`. Di nuovo se non si specifica altro il comando assume l'azione `show` e

Oggetto	Significato
`address`	imposta l'indirizzo fisico (*MAC address*) dell'interfaccia.
`arp`	abilita o disabilita (seguito da `on` o `off`) l'uso del protocollo ARP (vedi sez. 7.6.4 sull'interfaccia.
`broadcast`	imposta l'indirizzo di *broadcast* del livello fisico.
`dev`	indica l'interfaccia su cui operare.
`down`	disattiva l'interfaccia.
`multicast`	abilita o disabilita (seguito da `on` o `off`) l'uso del *multicast* sull'interfaccia.
`mtu`	imposta il valore della *Maximum Transfer Unit* dell'interfaccia.
`txqlen`	modifica la lunghezza della coda di trasmissione dell'interfaccia.
`up`	attiva l'interfaccia.

Tabella 7.22: Principali direttive per `ip link`.

stampa l'elenco delle rotte nella tabella di *routing* "*principale*", un esempio del risultato è il seguente:

```
root@gont:~# ip route
default via 192.168.168.1 dev eth0
192.168.168.0/24 dev eth0  proto kernel  scope link  src 192.168.168.10
```

E già qui iniziano ad emergere le differenze rispetto alla gestione illustrata in sez. 7.3.2: infatti con `route` si è sempre parlato di tabella di *routing* in maniera generica, e adesso ci si può legittimamente chiedere cosa significhi "*principale*". Il fatto è che in realtà in sez. 7.3.2 abbiamo fatto una semplificazione drastica, corrispondente alla situazione precedente la creazione dell'infrastruttura per il *routing* avanzato (il cosiddetto *advanced routing* di *iproute2*) quando è stata introdotta la possibilità di avere più tabelle di *routing*.

Nei kernel recenti infatti sono disponibili ben 255 tabelle di *routing*, le cui ultime tre (contraddistinte dai valori numerici 253, 254 e 255) sono riservate ed hanno un ruolo speciale. L'ultima, la 255, viene gestita direttamente dal kernel che, tutte le volte che questi vengono aggiunti con `ip address`, vi crea automaticamente le rotte relative agli indirizzi locali ed agli indirizzi di *broadcast*. Per questo motivo viene denominata `local`, e benché sia possibile farlo, non c'è nessun motivo di modificarla.

La tabella di *routing* principale, denominata `main`, è invece la 254; è qui che vengono inserite le rotte impostate con `route` nella gestione "*tradizionale*" degli indirizzi. Infine la terza tabella riservata è la 253, denominata `default`, che di norma è vuota ed è stata destinata, come indica il nome, a mettere le rotte da usare se non si è avuta corrispondenza in nessuna delle altre tabelle. Dato che la configurazione ordinaria (che va sulla `main`) comporta usualmente una rotta di default, questa evenienza non accade mai, per cui in genere non ci sono motivi per utilizzarla.

Quando si usa `ip`, che può controllare tutta l'infrastruttura del *routing* avanzato di Linux, il comando fa sempre riferimento alla tabella `main`, ed è per questo che prima si è parlato di tabella di *routing* "*principale*"; è comunque possibile operare su una tabella qualsiasi specificandola con la direttiva `table`, cui far seguire il nome o il numero. I tre nomi citati sono predefiniti in `/etc/iproute2/rt_tables`, dove se ne possono aggiungere altri nella forma di un numero seguito da un nome separato da spazi, con la solita convenzione di usare il "`#`" come commento.

A parte `show`, già illustrata, le due azioni principali di `ip route` sono `add` e `del`, che consentono di aggiungere o rimuovere una rotta e prendono sostanzialmente gli stessi argomenti. Entrambe richiedono che si indichi la destinazione con un indirizzo di rete o di una macchina con la direttiva

`to`, che essendo il default si può omettere. Gli indirizzi di rete (IPv4 o IPv6) devono sempre essere indicati con la notazione CIDR, se non si specifica una estensione l'indirizzo verrà sempre considerato come l'indirizzo di una singola stazione. Esiste inoltre il valore riservato `default` usato per indicare tutta Internet (cioè `0/0` per IPv4 o `::/0` per IPv6).

Infine all'indirizzo di destinazione può essere preposto l'indicatore del tipo di rotta, identificato da una parola chiave. Il default, che indica una rotta ordinaria e normalmente si omette, è `unicast`; ma si possono usare i tipi `blackhole` per scartare i pacchetti silenziosamente o `unreachable` e `prohibit` per scartarli con un errore rispettivamente di rete irraggiungibile o proibita.[29] Ci sono altri tipi per la descrizione dei quali si rimanda alla documentazione del comando accessibile con `man ip-route`.

La direttiva che consente di indicare quale indirizzo utilizzare come *gateway* è invece `via`, ed in genere occorre anche indicare l'interfaccia da usare con `dev`. Ad esempio per impostare il *default gateway* con il comando `ip` si dovrà eseguire qualcosa del tipo:

```
ip route add default via 192.168.0.1 dev eth0
```

mentre per impostare il *gateway* per una sottorete si dovrà usare qualcosa del tipo:

```
ip route add 192.168.168.0/24 via 192.168.0.234 dev eth0
```

Una direttiva che inoltre può risultare molto utile quando si utilizzano più indirizzi IP sulla stessa macchina è `src`, che consente di indicare quale di essi deve essere usato come indirizzo sorgente per i pacchetti in uscita attraverso quella rotta. Il comando consente inoltre di impostare metriche, *Maximum Transfer Unit* e *Round Trip Time* iniziale dei pacchetti con le direttive `metric`, `mtu` e `rtt`; si sono riassunte in tab. 7.23 le direttive principali, e di nuovo si rimanda alla documentazione per i dettagli.

Oggetto	**Significato**
`dev`	indica l'interfaccia di uscita.
`metric`	indica la metrica della rotta.
`mtu`	indica la *Maximum Transfer Unit* della rotta.
`rtt`	indica *Round Trip Time* iniziale.
`src`	indica l'indirizzo sorgente da usare per quella rotta.
`table`	indica su quale tabella operare.
`to`	indica la destinazione, ed essendo il default è normalmente omesso.
`via`	indica il *gateway* da usare per una destinazione.

Tabella 7.23: Principali direttive per `ip route`.

Si possono infine cancellare tutte le rotte disponibili con l'azione `flush`, cui passare eventualmente l'indicazione di una interfaccia con `dev` per restringere la cancellazione a quest'ultima. Oltre alle azioni generiche di tab. 7.20, `ip route` supporta anche due comode azioni specifiche, `save` e `restore` che consentono rispettivamente di salvare su un file, e ripristinare dallo stesso, il contenuto di una tabella.

Visto che fino ad adesso, anche negli esempi di `ip route`, abbiamo sempre fatto riferimento alle rotte della tabella `main`, ci si può chiedere a cosa servono e come vengano utilizzate le altre

[29]questo provoca la emissione di due diversi tipi di pacchetti ICMP che notificano l'errore (di rete irraggiungibile o vietata) all'IP sorgente di un pacchetto che ha come destinazione detta rete.

tabelle. Per capirlo occorre però una visione più approfondita di come viene gestito l'instradamento con la nuova infrastruttura dell'*advanced routing*, e come entrino in gioco le singole tabelle per ottenere la decisione finale sulla strada che dovrà seguire un pacchetto. Infatti a differenza del *routing* tradizionale, in cui la decisione veniva presa esclusivamente in base alla destinazione di un pacchetto, con il nuovo *routing* avanzato è possibile inserire criteri che consentono di effettuare delle scelte in base anche agli altri parametri di un pacchetto come l'indirizzo sorgente, il *Type of Service* (TOS) del pacchetto IP,[30] o le marcature (i cosiddetti *firewall mark*) attivabili con il *netfilter* di Linux.[31]

La nuova infrastruttura prevede che a fianco delle tabelle sia disponibile, all'interno di quello che viene chiamato *Routing Policy Data Base* (RPDB), un insieme di regole che vengono ordinate per numero crescente da 0 a 32767. Ciascun regola è composta da un selettore, che consente di scegliere a quali pacchetti applicarla, e da un esito, che dipende dal tipo di regola, ma che normalmente consiste nell'indicazione di quale tabella di *routing* usare per instradarli. Ma oltre all'instradamento si possono impostare (con la direttiva `type`) regole per scartare i pacchetti, usando nomi analoghi a quelli già visti per i tipi di rotte di `ip route` (`prohibit`, `blackhole`, ecc.). Il tipo di default è `unicast` (che si può anche omettere, come la direttiva `type`) che indica come esito la richiesta di esaminare una tabella di instradamento.

Per decidere l'instradamento il kernel esamina tutte le regole in ordine crescente a partire dalla numero 0 (che per questo ha la massima priorità), e se trova una corrispondenza con il relativo selettore prova ad utilizzare le rotte nella tabella indicata dalla regola stessa (o effettua quant'altro previsto dal tipo di regola). Se nella tabella è presente una rotta valida per il pacchetto questa verrà immediatamente utilizzata e la decisione sul *routing* conclusa, altrimenti il kernel proseguirà la scansione andando ad esaminare la regola successiva.

All'avvio del sistema vengono preimpostate tre regole, che sono quelle che consentono l'instradamento tradizionale, queste possono essere stampate con il comando `ip rule`, che come i precedenti se non si scrive nulla sottintende l'azione `show`; il risultato in tal caso sarà:

```
root@hogen:~# ip rule
0:	from all lookup local
32766:	from all lookup main
32767:	from all lookup default
```

La regola 0, che ha la priorità più alta, è riservata e non può essere né cancellata né modificata. Il risultato del comando precedente ci dice che essa si applica ai pacchetti con un indirizzo sorgente qualunque (cioè a tutti) per i quali viene consultata la tabella `local`. Come accennato in precedenza questa è la tabella predefinita mantenuta dal kernel che contiene gli indirizzi assegnati alle interfacce e quelli di *broadcast* e la regola serve appunto a gestire l'instradamento dei pacchetti quando si va verso un proprio indirizzo IP o si esegue un *broadcast*.

Le altre due regole, che sono quelle con i due valori massimi, e quindi con la priorità più bassa, sono invece modificabili e cancellabili, anche se normalmente non c'è alcun motivo per farlo. Di nuovo vengono selezionati tutti i pacchetti per i quali si consulta prima la tabella di *routing* principale (`main`) e poi quella con gli eventuali default (`default`). Per quanto detto

[30] si tratta di un campo dedicato presente in tutti i pacchetti IP, che indica, con un valore numerico di 8 bit, una tipologia di traffico a cui il pacchetto appartiene; viene appunto usato dai router per classificare i pacchetti prendere decisioni al loro riguardo.

[31] questa funzionalità, utilizzabile con il comando `iptables`, viene trattata in dettaglio, insieme a tutto quello che riguarda il *netfilter*, nel terzo capitolo di [SGL].

in precedenza questo significa che se non si sono aggiunte altre regole verranno usate le rotte impostate su `main` con i comandi tradizionali (o con `ip route` senza indicare una tabella) e la cosa si chiuderà lì, visto che normalmente `default` è vuota e comunque verrebbe consultata solo se non viene trovata una rotta su `main`.

La gestione avanzata del *routing* viene fatta pertanto aggiungendo, con `add`, ulteriori regole ad una priorità più elevata di queste ultime due (vale a dire con un valore numerico inferiore). La priorità può essere indicata esplicitamente con la direttiva `priority` seguita da un numero, mentre se non si specifica nulla viene utilizzato il valore numerico immediatamente precedente a quello più basso già presente. In sostanza di default comando assume che la regola che si immette sia a priorità più alta rispetto a quelle già presenti.

Inserendo una nuova regola in questo modo si può fare in modo che i pacchetti selezionati dalla stessa vengano instradati secondo le rotte inserite in una tabella creata ad hoc, da indicare con la direttiva `table` (o col sinonimo `lookup`). La gran parte delle direttive attiene alle modalità con cui si possono selezionare i pacchetti, si può allora usare `from` per indicare una selezione sugli indirizzi sorgente, `to` per selezionare sugli indirizzi di destinazione, `tos` per usare il campo TOS dei pacchetti, e `fwmark` per usare una etichetta del *netfilter* (il *firewall mark*); si sono elencate le principali direttive di selezione nella seconda parte di tab. 7.24.

Oggetto	**Significato**
`lookup`	sinonimo di `table`.
`priority`	indica la priorità della regola (con un valore numerico).
`table`	indica la tabella di instradamento da usare per i pacchetti selezionati dalla regola, si può specificare o il valore numerico o un nome configurato in `/etc/iproute2/rt_tables`.
`type`	indica il tipo di regola, il default è `unicast` ed in genere si omette completamente, altri valori possibili sono `blackhole`, `unreachable` e `prohibit` che consentono di scartare in vari modi i pacchetti selezionati dalla regola.
`from`	indica una selezione per indirizzo sorgente, sia di una macchina singola che di una sottorete.
`fwmark`	indica una selezione per il valore di un *firewall mark* impostato dal *netfilter*.
`iif`	indica una selezione per l'interfaccia di ingresso di un pacchetto.
`oif`	indica una selezione per l'interfaccia di uscita di un pacchetto (è disponibile solo per i pacchetti generati sulla macchina stessa).
`to`	indica una selezione indirizzo di destinazione.
`tos`	indica una selezione per il valore del campo TOS dei pacchetti IP.

Tabella 7.24: Principali direttive per `ip rule`.

Per capire meglio il funzionamento del comando ipotizziamo di avere, nella situazione di fig. 7.4, un secondo *router* con indirizzo `192.168.0.2`. Sulla macchina `havnor` si può allora impostare una regola che indichi di usarlo per le macchine della rete `192.168.168.0/24`. Per quanto appena detto, l'operazione deve essere compiuta in due passi, il primo è quello di selezionare tutti questi pacchetti ed indicare una tabella ad hoc per la loro gestione, questo si potrà fare con il comando:

```
ip rule add from 192.168.168.0/24 table 100 prio 1000
```

e si otterrà come risultato che:

```
root@havnor:~# ip rule
0:       from all lookup local
```

```
1000:        from 192.168.168.0/24 lookup 100
32766:        from all lookup main
32767:        from all lookup default
```

a questo punto si tratta di indicare per quei pacchetti un diverso *default gateway* usando la nuova tabella di *routing*, questo si fa semplicemente con:

```
ip route add default via 192.168.0.2 dev eth0 table 100
```

7.4 Il sistema della risoluzione dei nomi

Nella sezione precedente abbiamo trattato la configurazione di basso livello della rete, relativa alla parte di protocolli gestita direttamente dal kernel. Una delle funzionalità di base di Internet però è quella che consente di individuare un nodo sulla base di un nome simbolico invece che tramite il suo indirizzo IP. Esamineremo in questa sezione la configurazione del servizio di risoluzione dei nomi, che è quello che consente di fare tutto ciò. Vedremo come questa funzionalità sia fornita direttamente dalle librerie del sistema e quali sono i file di configurazione che ne controllano il comportamento.

7.4.1 Introduzione al *resolver*

Benché i protocolli di rete funzionino esclusivamente ad essi, è piuttosto raro che quando si intende utilizzare un servizio su Internet è piuttosto si debba fornire il valore numerico di un indirizzo IP. Questo avviene perché uno dei servizi di base per il funzionamento della rete è quello della risoluzione dei nomi, grazie al quale è possibile identificare un sito o un qualunque nodo della rete attraverso un nome simbolico che poi sarà *risolto* nel relativo indirizzo IP.

I nomi dei nodi di Internet, i cosiddetti *nomi a dominio*, vengono indicati con una forma specifica, come `nome.dominio.it`, che prevede la possibilità di raggruppare tutti i nodi che fanno capo ad una stessa organizzazione sotto un suffisso comune, detto appunto *dominio* (tratteremo i dettagli di tutto questo in sez. 9.1.1); nel caso in esempio il dominio è `dominino.it` mentre `nome` è il nome del nodo. Il nome del nodo viene normalmente viene chiamato *hostname*, anche se in realtà l'*hostname* è più precisamente il nome di una macchina in quanto tale, esso infatti è comunque definito, indipendentemente dal fatto che essa sia parte di una rete o di un dominio, anche se di solito, quando la si mette in rete, viene usato anche come nome del nodo.

Su Internet la risoluzione dei nomi viene realizzata attraverso un apposito servizio, il cosiddetto *Domain Name Service* (da qui in avanti DNS) che tratteremo più approfonditamente nel cap. 9. Questo servizio è in sostanza un enorme database distribuito, interrogabile via rete, che associa un nome simbolico (nella forma di un nome a dominio) al corrispondente indirizzo IP (in realtà il servizio DNS fornisce anche molte altre informazioni, ma torneremo su questo in sez. 9.1.2). Qualora sia necessaria una risoluzione si può interrogare il DNS; ogni programma che vuole accedere ad un nodo attraverso il suo nome invece che con il suo indirizzo IP è quindi un client per questo servizio.

Dato che la risoluzione dei nomi è un servizio usato da un gran numero di programmi diversi, in un sistema unix-like questo viene effettuato tramite una serie di funzioni, inserite direttamente nelle librerie standard del C, che vanno a costituire quell'insieme di funzionalità che viene

chiamato *resolver* e che permette di ottenere le associazioni fra nodi della rete ed i nomi identificativi associati agli stessi. Si tenga presente che tutto ciò non riguarda soltanto l'associazione fra nome a dominio e indirizzo IP, ma anche quella fra numero di porta e nome del corrispondente servizio ed altro ancora.

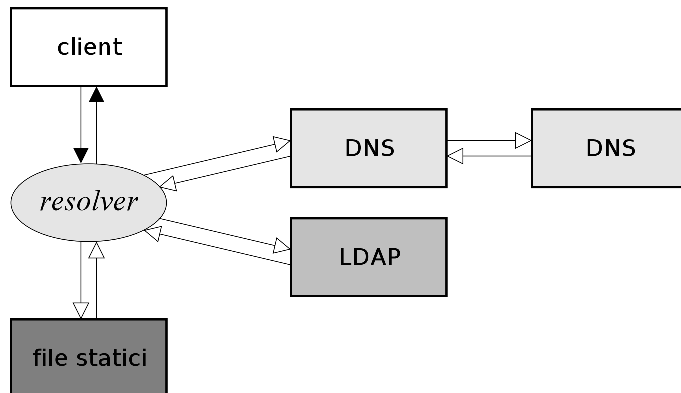


Figura 7.5: Schema di funzionamento delle funzioni del *resolver*.

Per questo prima di trattare del DNS in quanto servizio fornito sulla rete, dovremo prendere in considerazione i file di configurazione usati dalle funzioni di libreria del *resolver*, che sono quelle che svolgono in maniera generica il servizio di risoluzione dei nomi, e che non necessariamente eseguono questo compito rivolgendosi ad un server DNS. Le librerie infatti consentono di mantenere le informazioni relative alla risoluzione su diversi supporti, definendo una interfaccia astratta che viene usata dai programmi, secondo l'architettura illustrata in fig. 7.5.

Se restiamo nella nostra analogia telefonica, il sistema del *resolver* è in sostanza il meccanismo che ci porta dal conoscere il nome del destinatario all'avere il suo numero di telefono. Come per il telefono questo può avvenire in più modi, si può andare a cercare sulla rubrica, che nel caso è realizzata dal file `/etc/hosts` (vedi sez. 7.4.3), si può cercare nell'elenco del telefono, oppure ci si può rivolgere ad un server DNS, che è più simile al servizio "12" cui si chiedevano i numeri degli abbonati.[32] L'uso del DNS ha anche il vantaggio (come il servizio 12 rispetto ad una rubrica) che qualora una associazione fra nome e indirizzo IP dovesse cambiare questa viene aggiornata automaticamente.

Per questo motivo una delle informazioni che di norma vengono richieste nella configurazione manuale della rete è l'indirizzo IP del server DNS (ogni provider ne mette a disposizione almeno uno) che dovrà essere usato per ottenere la risoluzione dei nomi consentendo così la normale navigazione su Internet in cui si contattano i server con il loro nome a dominio. Ogni distribuzione prevede un suo sistema, e molte volte queste informazioni sono richieste anche in fase di installazione.

[32]anche se oggi non esiste più, continueremo a fare riferimento al vecchio numero, così da evitare pubblicità ad uno degli attuali fornitori di servizio, che ne se ne son già fatta fin troppa da soli.

7.4.2 I file di configurazione del *resolver*

Abbiamo già visto come in un sistema unix-like, oltre alla risoluzione dei nomi a dominio, esista la necessità di poter effettuare molte altre associazioni fra valori simbolici e numerici, come quelle fra numero di porta e nome del servizio, fra *user ID* e nome di login dell'utente, ecc. Siccome questo tipo di corrispondenze possono essere mantenute in diversi modi, le librerie del C del progetto GNU prevedono una modularizzazione di questi servizi, attraverso il sistema del *Name Service Switch*, che abbiamo trattato in sez. 4.3.6.

Come avevamo accennato allora questo sistema si applica anche ai nomi relativi alla rete, per i quali sono state previste una opportuna serie di classi; un estratto della sezione del file `/etc/nsswitch.conf` che concerne queste configurazioni è il seguente:

/etc/nsswitch.conf
```
...
hosts:          files dns
networks:       files
protocols:      db files
services:       db files
ethers:         db files
rpc:            db files
...
```

Nel caso di risoluzione dei nomi a dominio la riga che interessa è quella relativa a `hosts` (torneremo sulle altre in sez. 7.4.4). L'impostazione riportata nel precedente esempio ci dice che prima si dovrà andare a risolvere i nomi usando i file locali standard (cioè `/etc/hosts`) e poi il DNS. Se si volessero mettere le corrispondenze fra macchine locali e nomi su un server LDAP si potrebbe modificare questa linea come:

/etc/nsswitch.conf
```
hosts:          files ldap dns
```

nel qual caso, prima di interrogare il DNS, il *Name Service Switch* provvederebbe ad effettuare una ricerca su LDAP. A meno di non avere una esigenza specifica di questo tipo, in genere non ci sono motivi per modificare il contenuto di questo file.

Il sistema del *resolver* però è antecedente all'introduzione del *Name Service Switch* ed a lungo è stato controllato da una sua specifica serie di file di configurazione che consentivano, come vedremo, anche di scegliere l'ordine in cui viene effettuata la ricerca dei dati sui vari supporti per effettuare la risoluzione. Dato che la configurazione classica consentiva solo la scelta fra i file locali ed il DNS questo aspetto oggi viene gestito dal *Name Service Switch*, che consente una scelta su un numero arbitrario di supporti diversi; gli altri file vengono comunque utilizzati e sono quelli che permettono di controllare le modalità con cui si effettuano le interrogazioni al servizio DNS.

Il principale file di configurazione del *resolver* è `/etc/resolv.conf`, su cui vengono mantenute le informazioni di base necessarie al funzionamento dello stesso. Il file contiene la lista degli indirizzi IP dei server DNS a cui ci si rivolge quando si vuole effettuare una risoluzione e l'indicazione del dominio locale. Il formato del file è molto semplice, una serie di righe nella forma di una direttiva seguita da un valore (separato da spazi) con le righe vuote e quelle che iniziano per "`#`" che vengono ignorate. Un possibile esempio del suo contenuto è il seguente:

/etc/resolv.conf
```
search truelite.it
nameserver 127.0.0.1
nameserver 62.48.34.25
```

La direttiva `nameserver` serve ad indicare l'indirizzo IP di un server DNS a cui rivolgersi per la risoluzione dei nomi a dominio. Se ne possono specificare più di uno e la risoluzione di un nome verrà eseguita interrogandoli in sequenza, nell'ordine in cui essi sono stati specificati. Si possono indicare fino ad un massimo di 3 diversi server e l'indirizzo deve sempre essere fornito in forma numerica, anche qualora si disponesse di una risoluzione statica su `/etc/hosts`, dato che il *resolver* non può risolvere un nome senza aver prima interpretato correttamente il contenuto di questo file.

La direttiva `search` specifica una lista (separata da spazi) di domini in cui cercare i nomi delle macchine quando sono indicati in forma non qualificata.[33] Di norma si specifica come argomento solo il dominio locale, così quando si effettua la ricerca su un nome, gli viene aggiunto automaticamente quel dominio come suffisso. In questo modo si può specificare solo l'*hostname* di un computer sulla rete locale, e la ricerca verrà eseguita automaticamente come se lo si fosse scritto con il nome completo. Il tutto, se se ne è indicato più d'uno, verrà ripetuto per ciascuno dei domini presenti nella lista.

Il proprio dominio locale può comunque anche essere indicato direttamente con la direttiva `domain`, che indica il dominio in cui vengono cercati i nomi espressi in forma non qualificata, cui viene automaticamente aggiunto il dominio così specificato. In genere questa direttiva è alternativa alla precedente. Un elenco delle principali direttive usate nel file è riportato in tab. 7.25; una descrizione più dettagliata e completa si può trovare nella pagina di manuale, ottenibile con `man resolv.conf`.

Direttiva	**Significato**
`nameserver`	definisce l'IP di un server DNS da utilizzare per la successiva risoluzione dei nomi.
`search`	lista dei domini su cui effettuare una ricerca preventiva.
`domain`	nome del dominio locale, se assente viene determinato sulla base di quanto specificato con `/etc/hostname`.

Tabella 7.25: Direttive del file `resolv.conf`.

In genere le informazioni mantenute su `/etc/resolv.conf`, come il dominio locale e l'IP dei server DNS, vengono fornite in fase di installazione o di configurazione iniziale della rete. In sostanza è in questo file che viene indicato qual'è il numero di telefono del servizio "12" a cui ci si rivolge; ogni provider ne mette a disposizione almeno uno, e come vedremo nel cap. 9 è possibile anche installarsi un server DNS locale.

Per una macchina posta in una rete locale questi valori di norma devono essere impostati manualmente, a meno di non usare il servizio DHCP (vedi sez. 8.2.2) per l'impostazione automatica. In genere questo viene fatto dal programma di configurazione della rete, che, quando vi chiede dominio e DNS, va a creare automaticamente questo file, inserendovi le direttive `search` e `nameserver` con i valori da voi forniti.

[33] per forma qualificata di un nome a dominio si intende un nome completo del tipo `freedom.softwarelibero.it`, usare una forma non qualificata all'interno del dominio `softwarelibero.it` sarebbe usare semplicemente `freedom`.

Se invece si sta usando un collegamento punto-punto con un modem o una ADSL sono i programmi che lanciano la connessione (cioè `pppd` o l'eventuale interfaccia a quest'ultimo, vedi sez. 7.5.2) che usano le informazioni ottenute dal provider durante la fase di negoziazione del collegamento, e riscrivono al volo questo file; in questo caso può servire aggiungere al volo in un secondo tempo qualche altro DNS (ce ne sono di pubblici) qualora quello del provider avesse problemi.

Il secondo file di configurazione principale del *resolver* è `/etc/host.conf`, che controlla le modalità di funzionamento delle funzioni che eseguono la risoluzione dei nomi. Al solito le righe vuote od inizianti per "`#`" vengono ignorate, le altre devono contenere una parola chiave che esprime una direttiva, che può essere seguita o meno da un valore. Un esempio comune del contenuto di questo file è il seguente:

/etc/host.conf
```
order hosts,bind
multi on
```

La direttiva `order` controlla la sequenza in cui viene effettuata la risoluzione dei nomi, nell'esempio si dice che deve prima essere usato il file `/etc/hosts` (vedi sez. 7.4.3), e poi possono essere eseguite eventuali interrogazioni ai DNS esterni. Ad oggi questa parte viene gestita, come accennato, dal *Name Service Switch*, ma si abbia cura, per evitare ambiguità, di usare una configurazione coerente, oppure si eviti di impostarla.

La direttiva `multi` permette di ottenere come risposta tutti gli indirizzi validi di una stazione che compare più volte in `/etc/hosts`, invece di avere solo il primo, non conviene attivarla nel caso di che questo file risulti molto voluminoso, in quanto comporta un impatto negativo sulle prestazioni.

Oltre a queste due le altri principali direttive (ed il relativo significato) sono riportate in tab. 7.26. Tutte prendono come argomenti i valori `on` ed `off`, che attivano e disattivano il comportamento richiesto. Quando non vengono esplicitamente utilizzate, il comportamento di default è equivalente ad `off`. Per l'elenco completo ed i dettagli su tutte le direttive si può al solito consultare la pagina di manuale con `man host.conf`.

Direttiva	**Significato**
`multi`	ritorna tutti gli indirizzi corrispondenti presenti in `/etc/hosts` e non solo il primo, prende i valori `on` o `off` (il default).
`nospoof`	attiva un controllo *antispoofing*[34]elementare chiedendo la risoluzione inversa dell'IP ricevuto e fallendo in caso di mancata corrispondenza, prende i valori `on` o `off` (il default).
`order`	indica come ed in che ordine viene eseguita una ricerca, prevede i valori `bind`, `hosts` e `nis` da indicare separati da virgole se se ne specifica più di uno.
`reorder`	riordina gli indirizzi in modo da restituire per primi quelli locali, prende i valori `on` o `off` (il default).
`spoofalert`	se `nospoof` è attivo inserisce un avviso degli errori rilevati nei log di sistema, prende i valori `on` o `off` (il default).

Tabella 7.26: Principali direttive del file `host.conf`.

[34]con *spoofing* si intende in genere quella tecnica che prevede la creazione di pacchetti IP con indirizzo sorgente falsificato, per cercare di impersonare una macchina diversa.

7.4.3 La gestione locale dei nomi

Normalmente il primo metodo utilizzato dal *resolver* per effettuare una corrispondenza fra nomi simbolici e indirizzi IP è utilizzare il contenuto di `/etc/hosts`. Questo file contiene un elenco di nomi di macchine, associati al relativo indirizzo IP. Lo si usa quindi come un'agenda del telefono per specificare gli indirizzi delle macchine per le quali si ha una mappa *statica* degli indirizzi, ad esempio le macchine di una rete privata che non vanno su Internet, ma che volete risolvere col nome che gli avete assegnato.

Il formato del file è molto semplice, al solito le righe vuote od inizianti per "`#`" sono ignorate mentre le altre righe devono contenere, separati da spazi o caratteri di tabulazione, l'indirizzo IP, il nome completo (in termini di dominio, quello che viene detto FQDN, *Fully Qualified Domain Name*, vedi sez. 9.1.1) ed una eventuale lista di nomi alternativi. Un possibile esempio di questo file è il seguente:

/etc/hosts
```
127.0.0.1       localhost
192.168.168.1   havnor.gnulinux.it     havnor

# private nets
192.168.168.10  gont.gnulinux.it       gont
192.168.168.11  oppish.gnulinux.it     oppish

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

che associa agli IP delle macchine che in fig. 7.4 sono sulla rete secondaria di `havnor` i relativi nomi.

Ovviamente usare questo file è la maniera più semplice per identificare una macchina su una rete locale, lo svantaggio è che deve essere presente su ogni macchina, e quando il numero di nodi coinvolti aumenta, diventa sempre più complicato il lavoro di tenerli tutti aggiornati e coerenti fra loro. Per questo vedremo in sez. 9.2.4 come fare lo stesso lavoro attraverso un server DNS locale.

Benché il file faccia riferimento a delle caratteristiche tipiche dell'uso della rete, esso deve essere presente anche quando la macchina non è fisicamente in rete perché l'interfaccia di *loopback* viene comunque utilizzata da molti programmi, che fanno riferimento a questo file per la risoluzione del nome `localhost`, ad essa associato, sull'indirizzo `127.0.0.1`.

Un caso particolare di gestione dei nomi è quello che riguarda il nome che contraddistingue la macchina stessa, comunemente detto *hostname*. Benché l'*hostname* abbia di solito più significato quando si è collegati in rete, come accennato il nome di una macchina è una proprietà del tutto indipendente dalla presenza in rete della stessa, e viene riportato anche da vari comandi di sistema non attinenti alla rete (come `uname`) nel prompt della shell o nelle schermate di avvio.

Di norma buona parte delle distribuzioni chiedono il nome della macchina in fase di installazione e lo memorizzano in un opportuno file di configurazione: nel caso di Debian il nome viene semplicemente scritto nel file `/etc/hostname`, RedHat usa la variabile `HOSTNAME` nel file

/etc/sysconfig/network, altre distribuzioni possono fare scelte diverse. Questo file viene usato negli script di avvio per leggere il nome della macchina quando questo viene impostato con il comando hostname o utilizzato direttamente nell'avvio con systemd. Perciò si deve essere consapevoli che cambiare il contenuto del file su cui è memorizzato non cambia il nome della macchina fintanto che non si chiama direttamente anche hostname o si rieseguono gli script di avvio.

Se invocato direttamente il comando hostname si limita a stampare il nome della macchina, mentre può impostare un nome diverso se quest'ultimo viene passato come argomento, se invece si usa l'opzione -F il comando legge il nome da un file (ad esempio è con questa opzione che gli script di avvio di Debian usano il contenuto di /etc/hostname). Si può richiedere anche la stampa del nome a dominio completo con l'opzione --fqdn, questo però dipende dal fatto che si sia configurato il *resolver* per effettuare la risoluzione dei nomi, e non può essere modificato dal comando stesso.[35]

Il comando può essere invocato anche come dnsdomainname che invece permette di ottenere il nome del dominio di cui si fa parte. Anche in questo caso non è possibile modificarne il valore con il comando, in quanto di nuovo questo valore viene sempre determinato tramite il *resolver* sulla base della configurazione di quest'ultimo. Per le altre opzioni del comando ed per una descrizione completa si può fare riferimento alla pagina di manuale, accessibile con man hostname.

7.4.4 La gestione degli altri nomi di rete

Come accennato in sez. 7.4.2 oltre alle associazioni fra indirizzo IP e nome simbolico di una macchina, esistono una serie di altre informazioni (si faccia riferimento a quanto riportato in tab. 4.29) relative a protocolli e servizi di rete, di norma identificate da valori numerici, ai quali poi viene assegnato un corrispondente nome simbolico come facilitazione mnemonica.

Queste corrispondenze oggi sono gestite tutte tramite il *Name Service Switch*, ma quasi sempre, come risulta dall'estratto di configurazione citato in sez. 7.4.2, vengono utilizzati i file statici su cui tradizionalmente[36] esse venivano mantenute, per questo motivo nel resto della sezione tratteremo soltanto del formato di questi file.

In sez. 7.2.4 abbiamo già accennato come la corrispondenza fra i servizi di rete ed i numeri di porta loro assegnati venga mantenuta nel file /etc/services; è al contenuto di questo file che fanno riferimento tutti i programmi che vogliono utilizzare il nome simbolico di un servizio al posto del valore numerico della porta ad esso associato.

Di norma questo non è un file che sia necessario modificare, ma è utile conoscerne il contenuto. Il formato del file è molto semplice, un elenco di numeri di porta a ciascuno dei quali è associato un nome simbolico che individua il servizio ad esso associato dalle convenzioni internazionali o dall'uso comune. Un estratto di questo file, preso da una Debian Squeeze, è il seguente:

/etc/services
```
...
ftp-data        20/tcp
```

[35] quello che succede è che il comando esegue la risoluzione del nome della macchina (ottenuto con hostname) sul *resolver* e riporta il risultato ottenuto; in genere detta risoluzione viene inserita in fase di installazione in /etc/hosts insieme all'IP della stessa, per cui di fatto in tal caso detto file contiene anche la configurazione del proprio dominio locale.

[36] cioè prima che si passasse all'uso del *Name Service Switch*, quando le informazioni erano appunto ricavate direttamente dalla scansione di questi file.

```
ftp             21/tcp
fsp             21/udp          fspd
ssh             22/tcp                          # SSH Remote Login Protocol
ssh             22/udp
telnet          23/tcp
smtp            25/tcp          mail
time            37/tcp          timserver
time            37/udp          timserver
rlp             39/udp          resource        # resource location
nameserver      42/tcp          name            # IEN 116
whois           43/tcp          nicname
tacacs          49/tcp                          # Login Host Protocol (TACACS)
tacacs          49/udp
re-mail-ck      50/tcp                          # Remote Mail Checking Protocol
re-mail-ck      50/udp
domain          53/tcp                          # name-domain server
domain          53/udp
mtp             57/tcp                          # deprecated
tacacs-ds       65/tcp                          # TACACS-Database Service
tacacs-ds       65/udp
bootps          67/tcp                          # BOOTP server
bootps          67/udp
bootpc          68/tcp                          # BOOTP client
bootpc          68/udp
tftp            69/udp
gopher          70/tcp                          # Internet Gopher
gopher          70/udp
rje             77/tcp          netrjs
finger          79/tcp
www             80/tcp          http            # WorldWideWeb HTTP
www             80/udp                          # HyperText Transfer Protocol
...
```

Come per buona parte dei file di configurazione, righe vuote e tutto quello che segue il carattere "#" viene considerato un commento ed ignorato; ogni riga ha il formato:

```
nome            numero/protocollo    alias
```

dove `nome` é l'identificativo simbolico del servizio, `numero` è il numero di porta ad esso assegnato, `protocollo` indica se il servizio usa UDP o TCP, mentre `alias` è una lista (separata da spazi) di eventuali altri nomi associati allo stesso servizio.

Un secondo file che mantiene corrispondenze fra valori numerici e nomi è `/etc/networks`, che è l'analogo di `/etc/hosts` per quanto riguarda le reti. Anche queste, come le singole stazioni, possono essere identificate da un nome, e di nuovo le corrispondenze statiche fra nome e indirizzo IP della rete vengono mantenute in questo file, il cui formato, come descritto dalla pagina di manuale accessibile con `man networks`, è composto da tre campi separati da spazi.

Il primo campo indica il nome simbolico della rete, il secondo campo il suo indirizzo, nella notazione *dotted decimal* (tralasciando opzionalmente gli eventuali `.0` finali), ed il terzo campo un eventuale alias (questo campo è opzionale). Al solito le righe vuote e tutto quello che segue un `#` viene ignorato. Un esempio del contenuto di questo file potrebbe essere:

/etc/networks

```
localnet 192.168.1.0
```

il contenuto di questo file viene utilizzato da comandi come `netstat` o `route` per mostrare i nomi simbolici al posto dei valori numerici; si tenga conto però che con questo file viene supportata solo la corrispondenza con reti espresse nella notazione tradizionale per classi di tipo A, B o C, e che la notazione in formato CIDR non funziona.

Un altro file contenente corrispondenze fra valori numerici e nomi simbolici è `/etc/protocols`, che associa il numero usato per indicare all'interno dei pacchetti IP il protocollo di trasporto usato nello strato successivo (si ricordi quanto detto in sez. 7.1.3) ad un nome identificativo del protocollo stesso (come TCP e UDP).

Il formato di questo file è identico a quello di `/etc/networks`; sono supportati tre campi divisi da spazi, in cui il primo identifica il nome simbolico, il secondo il valore numerico che identifica il protocollo ed il terzo un alias. Un esempio di questo file è il seguente (l'estratto è preso da una Debian):

/etc/protocols

```
...
ip      0       IP              # internet protocol, pseudo protocol number
#hopopt 0       HOPOPT          # IPv6 Hop-by-Hop Option [RFC1883]
icmp    1       ICMP            # internet control message protocol
igmp    2       IGMP            # Internet Group Management
ggp     3       GGP             # gateway-gateway protocol
ipencap 4       IP-ENCAP        # IP encapsulated in IP (officially ''IP'')
st      5       ST              # ST datagram mode
tcp     6       TCP             # transmission control protocol
egp     8       EGP             # exterior gateway protocol
igp     9       IGP             # any private interior gateway (Cisco)
pup     12      PUP             # PARC universal packet protocol
udp     17      UDP             # user datagram protocol
...
```

Al solito le righe vuote e tutto quello che segue un carattere "`#`" viene ignorato; come per tutti gli altri file analoghi la descrizione completa del formato è riportata nella rispettiva pagina di manuale, accessibile con `man protocols`.

7.5 Altre tipologie di connessioni di rete

Finora abbiamo trattato delle configurazioni di rete facendo riferimento alla casistica tipica di una rete locale basata sulle ordinarie schede Ethernet; in questa sezione esamineremo invece la configurazione della rete per altre tecnologie di connessione, come le interfacce punto-punto che si usano per collegarsi tramite un modem o le interfacce *wireless* che si usano per collegarsi con un *access-point*.

7.5.1 Cenni sul protocollo PPP

Per le configurazioni di base illustrate in sez. 7.3 abbiamo sempre fatto riferimento al caso di una macchina inserita all'interno di una rete locale, con il caso più comune dell'uso di schede Ethernet, per cui ad una interfaccia non viene associato solo un indirizzo IP, ma anche una maschera di rete.

In generale però questo non è vero, ed è infatti possibile collegarsi ad Internet attraverso un modem, analogico o ADSL che sia. In tal caso la comunicazione verso l'esterno non avviene attraverso il passaggio per una rete locale, ma direttamente con una macchina posta all'altro capo della connessione, con quello che si chiama un collegamento *punto-punto*.

In sez. 7.2.1 abbiamo visto che esiste un apposito protocollo del livello di collegamento fisico, il *Point to Point Protocol* (comunemente detto PPP) su cui è possibile innestare tutti i protocolli dei livelli successivi. Nel caso specifico PPP è un protocollo generico che permette di inviare dati su diverse tipologie di collegamenti fisici, mantenendo una stessa struttura generale, che consente di creare e gestire connessioni punto-punto.

Il protocollo si compone di tre parti, un metodo generico per incapsulare dati su un collegamento fisico, un protocollo estensibile per il controllo del collegamento fisico (il *Link Control Protocol* o LCP) ed una famiglia di protocolli (il *Network Control Protocols* o NCP) per stabilire e configurare le connessioni coi protocolli di livello superiore.

In generale PPP fa riferimento ad un qualche meccanismo sottostante di trasmissione (che sia un modem analogico, ISDN o ADSL) sopra il quale costruisce uno strato ulteriore che gli permette di incapsulare i protocolli di livello superiore. L'uso più tipico di questo protocollo è comunque quello che viene fatto con i modem analogici o ADSL, per i quali permette di gestire la creazione della connessione verso il fornitore del servizio di connessione e la configurazione della relativa interfaccia.

7.5.2 Il demone `pppd`

Delle tre parti illustrate in sez. 7.5.1 di cui è composto il protocollo PPP, l'incapsulazione dei dati è gestita direttamente dal kernel. Il relativo supporto è abilitato di default in tutte le distribuzioni, ed in genere da questo punto di vista non c'è da fare niente, se non assicurarsi, qualora si sia ricompilato il proprio kernel (vedi sez. 5.2.3) di aver incluso quanto necessario nella sezione *Network device support*. Tutto il resto è affidato ad un apposito demone, `pppd`, che si cura di fornire il controllo del collegamento, l'autenticazione, e la parte dell'NCP che riguarda la configurazione e la gestione del protocollo IP su PPP, compresa l'impostazione di una interfaccia di rete e delle eventuali rotte ad essa associate.

Il programma può essere lanciato direttamente dalla riga di comando passandogli come argomenti le varie direttive che ne controllano il comportamento, ma in genere l'invocazione avviene attraverso un opportuna interfaccia che consente di passargli le opzioni di configurazione.[37] Di default infatti, se invocato senza nessun argomento, le direttive vengono lette all'avvio del demone dal file `/etc/ppp/options`. Le direttive, nella documentazione del programma, vengono chiamate *opzioni*, qui le chiamiamo così per evitare confusione con le normali opzioni dei comandi ordinari.

Alternativamente si può specificare un singolo argomento che indichi il dispositivo da usare per la connessione. Questo deve essere indicato con il relativo file di dispositivo, che in genere è la seriale a cui è agganciato il modem, espresso in forma di pathname sia assoluto che relativo rispetto a `/dev`. In questo caso le ulteriori direttive saranno lette dal file `/etc/ppp/options.`*`ttyname`*, ed è così possibile scegliere automaticamente configurazioni diverse per i diversi dispositivi; se ad esempio si usa il comando:

[37] abbiamo ad esempio accennato in sez. 7.3.3 come con Debian questo possa essere fatto direttamente da `/etc/network/interfaces`.

```
pppd /dev/ttyS0
```

le direttive saranno lette dal file `options.ttyS0`.

Inoltre può risultare utile consentire anche agli utenti normali di attivare una connessione. In genere `pppd` è *suid* per root ed eseguibile per gli utenti che fanno parte di un opportuno gruppo (`dip` nel caso di Debian). Per questo motivo è prevista una forma diversa di avvio del demone eseguita utilizzando la direttiva `call` a cui far seguire un nome che indichi il fornitore di servizi (il *provider*) che si vuole utilizzare.

In questa forma infatti l'uso del comando è consentito anche ad un utente normale e le direttive saranno lette da un file con quel nome posto nella directory `/etc/ppp/peers`.[38] In realtà un utente potrebbe eseguire comunque il comando, nel qual caso esso farebbe riferimento alle direttive presenti nel file `.ppprc` della sua home, ma questo file non può contenere direttive privilegiate ed è pertanto sottoposto a restrizioni che lo rendono poco pratico (per i dettagli si consulti la pagina di manuale). Così se ad esempio `/etc/ppp/peers/adsl` contiene le opzioni per chiamare il proprio provider con un modem ADSL, potremo attivare la connessione con:

```
pppd call adsl
```

Una volta lanciato `pppd` esegue le operazioni necessarie a stabilire una connessione. Le modalità con cui questo avviene dipendono però dal tipo di linea che si usa, ad esempio se si ha un modem analogico agganciato al telefono occorrerà effettuare una chiamata telefonica ed ottenere la linea prima che `pppd` possa entrare in azione. Essendo possibili diverse modalità la eventuale preparazione della linea viene sempre delegata ad un programma esterno con l'uso della direttiva `connect`; questa richiede come argomento una riga di comando che verrà eseguita in una shell, fintanto che il comando non sarà eseguito con successo, `pppd` non farà alcun accesso alla linea.

Nel caso di collegamenti telefonici tramite modem analogico per la preparazione della linea si fa normalmente ricorso all'uso del comando `chat`, ed un esempio possibile della sua invocazione potrebbe essere tramite una direttiva del tipo:

```
connect "/usr/sbin/chat -v -f /etc/chatscripts/provider"
```

In questo caso `chat` viene usato per dare le istruzioni al modem e verificare la riuscita della chiamata. Le opzioni più importanti del comando sono `-f` che permette di indicare un file che contiene le istruzioni da eseguire, e `-v` che permette di registrare i risultati della comunicazione a scopo di controllo. Il comando opera leggendo dal file indicato (il cosiddetto "*chat script*") le istruzioni con le quali gestire la comunicazione iniziale con il modem.

Uno script di chat è usualmente composto da una serie di coppie di valori, in cui il primo valore indica quello che ci si attende di sentire dal dispositivo ed il secondo quanto si deve scrivere in risposta. Le coppie si scrivono usualmente una per riga in due campi separati da spazi per maggiore leggibilità ma si possono scrivere anche spezzate su più righe (nel qual caso il ritorno a capo vale come separatore) o su una sola riga; questo comporta che se si vuole aspettare o inviare un valore che contiene spazi occorrerà esprimerlo scritto fra virgolette. Alla ricezione del valore specificato nel primo campo viene inviata la stringa indicata nel secondo,[39] in questo modo si può controllare il modem ed inviare comandi e dati sulla linea in una specifica sequenza.

[38] che è anche il metodo usato da Debian per fare eseguire il comando all'avvio con `/etc/network/interfaces`.

[39] il default è inviare la stringa seguita da un ritorno a capo, si può disabilitare questo comportamento aggiungendo il carattere di controllo "`\c`".

In realtà oltre alle normali coppie di valori di attesa/risposta, il comando supporta anche alcune istruzioni chiave; ad esempio con `ABORT` usato come valore di attesa si definiscono le condizioni in cui viene abortita la connessione, specificate nel valore di risposta con altrettante parole chiave, come `BUSY`, `VOICE`, `"NO DIALTONE"`, ecc. che valgono indipendentemente dal momento in cui la condizione si va a verificare nella sequenza delle istruzioni (per i valori possibili si consulti la pagina di manuale).

Esistono inoltre delle sequenze di controllo, introdotte dalla solita barra rovescia "`\`", che consentono di inserire caratteri speciali come il ritorno a capo, o, con l'espressione "`\T`", un valore arbitrario (in genere usato per indicare il numero di telefono), passato al comando `chat` con l'opzione `-T`. Al solito per i dettagli sulle opzioni e sulle altre istruzioni di controllo si consulti la pagina di manuale.

Un esempio elementare di *chat script*, estratto dal file `/etc/chatscripts/pap` che viene distribuito con Debian Lenny, è il seguente, dove si sono omesse le righe di commento, introdotte al solito dal carattere "`#`":

/etc/chatscripts/pap

```
ABORT            BUSY
ABORT            VOICE
ABORT            "NO CARRIER"
ABORT            "NO DIALTONE"
ABORT            "NO DIAL TONE"
""             ATZ
OK             ATDT\T
CONNECT            ""
```

Si noti come dopo aver definito le varie condizioni di interruzione, si inizia lo script non aspettandosi niente (questo il significato della stringa vuota `""`) a cui si risponde con il comando `ATZ` del modem.[40] Alla risposta `OK` si esegue il comando `ATDT`, che esegue la telefonata al numero telefonico indicato con l'opzione `-T` di `chat`. L'uso di `-T` è del tutto opzionale, si sarebbe potuto scrivere esplicitamente il numero di telefono all'interno dello script di chat. Alla risposta `CONNECT` lo script finisce e il controllo passa direttamente a `pppd` che a questo punto si suppone sia in grado di parlare con il corrispettivo dall'altra parte.

Si tenga presente che uno script siffatto non è detto sia sempre corretto, ad esempio alcuni provider richiedevano una procedura di login prima di attivare il proprio demone `pppd` sull'altro capo della connessione, questo poteva comportare la necessità di ulteriori linee come:

/etc/chatscripts/pap

```
ogin: username
ssword: password
```

in cui alla richiesta di `login:` viene inviato un username, ed alla successiva richiesta di `password:` viene inviata la password. Si noti come in quest'ultimo esempio a `chat` basti una corrispondenza parziale nella richiesta per inviare una risposta; la ragione di tutto ciò origina nei difetti presenti ai tempi delle prime connessioni via modem che talvolta provocavano la perdita dei primi caratteri inviati; usando una corrispondenza parziale la procedura poteva proseguire senza errori.

[40] non tratteremo qui i cosiddetti *Comandi Hayes* (o comandi AT) usati dai modem per gestire le linee telefoniche, per un riferimento si consulti `http://en.wikipedia.org/wiki/Hayes_command_set`.

Oggi le connessioni via modem analogico sono sempre meno comuni, ma la precedente trattazione resta valida per tutte le connessioni eseguite tramite linee seriali, come quelle associate a varie schede di comunicazione GSM o GPRS, o alle connessioni effettuate tramite emulazione di una seriale sul collegamento *bluetooth* al proprio cellulare.

La situazione è invece del tutto diversa se si ha a che fare con un modem ADSL, in tal caso infatti non si dovrà affatto effettuare una telefonata o la preparazione della linea, per cui non sarà più necessario invocare `chat`. In questo caso la configurazione varia a seconda del tipo di modem utilizzato e dal tipo di protocollo (principalmente PPPoA, più raro PPPoE) ed i dettagli eccedono quanto sia possibile trattare qui.

Se la connessione viene stabilita regolarmente, il demone `pppd` si incaricherà di creare una nuova interfaccia di comunicazione (il default è l'uso di `ppp0`) associata alla connessione ed opportunamente configurata in modalità punto-punto; si avrà cioè qualcosa del tipo:

```
anarres:~# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:8890912 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8890912 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3729080730 (3.4 GiB)  TX bytes:3729080730 (3.4 GiB)

ppp0      Link encap:Point-to-Point Protocol
          inet addr:151.49.5.174  P-t-P:151.6.144.72  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:9178  Metric:1
          RX packets:73874097 errors:0 dropped:0 overruns:0 frame:0
          TX packets:72996070 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:1180778093 (1.0 GiB)  TX bytes:3882710341 (3.6 GiB)
```

Se poi si è utilizzata la direttiva `defaultroute` il programma si incaricherà anche di impostare automaticamente una rotta di default per far passare dalla interfaccia il traffico diretto verso Internet,[41] mentre con la direttiva `usepeerdns` si richiede l'impostazione automatica dei DNS forniti dal provider con la creazione automatica di un opportuno `resolv.conf` all'attivazione del collegamento ed il ripristino del precedente alla sua conclusione.

Inoltre tutte le volte che l'attivazione di un collegamento ha successo viene lanciato lo script `/etc/ppp/ip-up` che si cura eseguire una serie di azioni programmate relative alla presenza di una nuova connessione (ad esempio far scaricare la posta). In generale questo script si limita ad eseguire tutti gli script posti in `/etc/ppp/ip-up.d` con il comando `run-parts` che abbiamo già visto in sez. 3.2.1. Allo stesso modo, quando la connessione viene chiusa viene lanciato lo script `/etc/ppp/ip-down` che esegue quelli presenti in `/etc/ppp/ip-down.d`, dopo di che l'interfaccia di rete viene disattivata.

In tab. 7.27 si sono riportate le direttive di configurazione più significative di `pppd`, insieme ad una loro breve descrizione. L'elenco completo, ed i dettagli relativi a ciascuna di esse sono al solito disponibili nella pagina di manuale del comando, accessibile con `man pppd`.

[41] come già spiegato in sez. 7.3.2 la creazione di una rotta per l'altro capo della connessione punto-punto avviene automaticamente alla configurazione dell'interfaccia.

Direttiva	**Significato**
`call` *`name`*	usa le opzioni contenute nel file *`name`* in `/etc/ppp/peers`.
`connect` *`script`*	usa lo script *`script`* (di norma `chat`) per preimpostare la linea.
`crtscts`	attiva il controllo di flusso hardware sulle porte seriali.
`defaultroute`	aggiunge una rotta di default alla tabella di instradamento usando l'indirizzo IP ricevuto sulla connessione.
`lock`	crea un file di lock per la seriale.
`demand`	crea la connessione a richiesta quando vede del traffico.
`usepeerdns`	chiede alla connessione gli indirizzi di due server DNS, che vengono passati allo script `/etc/ppp/ip-up` nelle variabili di ambiente `DNS1` e `DNS2` ed usati per la creazione di un file `resolv.conf` che li usi come server DNS.
`debug`	abilita il debugging, è utile per avere più informazioni quando la connessione non funziona.

Tabella 7.27: Principali opzioni per il demone `pppd`.

7.5.3 I meccanismi di autenticazione

Al di là della possibile necessità di autenticarsi per l'accesso direttamente sulla linea seriale (come mostrato nell'esempio riguardante `chat`) il demone `pppd` prevede una sua procedura di autenticazione diretta, disponibile quando stabilisce la comunicazione con il suo corrispettivo all'altro lato della connessione. Questa autenticazione può avvenire secondo due protocolli, il *Password Authentication Protocol* (noto come PAP) ed il *Challenge Handshake Authentication Protocol* (noto come CHAP).

Il PAP è un protocollo più elementare e prevede che il client invii un username ed una password in chiaro sulla linea; si è così esposti ad una eventuale intercettazione telefonica. Il CHAP invece prevede l'invio da parte del server di una *sfida* nella forma di un pacchetto che contiene il nome del server, così che il client può rispondere con un pacchetto che contenga un *hash* crittografico del pacchetto di sfida cui si è aggiunto il valore di un segreto condiviso (la password) così da comprovare la conoscenza del segreto senza inviarlo sulla linea.

Le informazioni di autenticazione usate dal demone `pppd` per i due protocolli vengono mantenute in due file, `/etc/ppp/pap-secrets` e `/etc/ppp/chap-secrets`, che hanno lo stesso formato. Un esempio del contenuto di uno questi file è il seguente:

/etc/ppp/pap-secrets
```
# INBOUND connections
# UserIDs that cannot use PPP at all. Check your /etc/passwd and add any
# other accounts that should not be able to use pppd!
guest   frijole "*"     -
master  frijole "*"     -
root    frijole "*"     -
support frijole "*"     -
stats   frijole "*"     -
# OUTBOUND connections
piccardi *       password
```

Il file prevede quattro campi separati da spazi; il primo campo indica l'username usato dal client, il secondo il nome del server, il terzo il segreto condiviso, il quarto, opzionale, una lista

degli IP da cui è possibile effettuare il collegamento (un "-" indica nessun IP), si possono indicare delle sotto-reti in notazione CIDR ed usare un "!" iniziale per negare la selezione.

I nomi di client e server possono contenere caratteri qualunque, ma gli spazi e gli asterischi devono essere protetti scrivendoli fra virgolette; il valore `"*"` indica un nome qualsiasi e fa da carattere jolly. Se il campo della password inizia con il carattere "`@`" si indica che quest'ultima deve essere letta dal file il cui nome è specificato dal resto della stringa.

Dato che il collegamento è punto-punto il procedimento di autenticazione è simmetrico: ogni demone può chiedere all'altro di autenticarsi; per questo i file contengono sia le informazioni per essere autenticati presso gli altri che per autenticarli. Di norma però, a meno di non gestire un provider, si deve solo essere autenticati, per questo il default è che `pppd` non esegue richieste di autenticazione, ma si limita a rispondere a quelle che gli vengono fatte.

7.5.4 La connettività *Wi-Fi*

Una delle tecnologie di rete con sempre maggiore diffusione, per il vantaggio di non richiedere la presenza di una infrastruttura cablata per realizzare la comunicazione, è quella legata alle tecnologie basate sulla trasmissione dati via radio, che viene indicata in maniera generica come *wireless*. In realtà le tecnologie *senza fili* sono molte ma quella che tratteremo in questa sezione e nelle successive è quella che viene più propriamente indicata come *Wi-Fi* (dal nome adottato dal relativo consorzio di produttori) e che fa riferimento allo standard di comunicazione IEEE 802.11 nelle sue numerose varianti.

A differenza di quanto visto per PPP in questo caso si tratta, come si può capire dall'uso di uno standard della stessa famiglia di quello delle classiche reti Ethernet cablate, di una tecnologia pensata per la realizzazione di reti locali; pertanto molti dei concetti già visti per quel protocollo di collegamento fisico (come il *MAC address*) si applicano anche a queste tecnologie.

La maggiore differenza con Ethernet in questo caso, a parte l'assenza di connessioni via cavo, è che il protocollo prevede a livello di collegamento fisico dei meccanismi di autenticazione e di cifratura dei dati, onde impedire che chiunque possa collegarsi ad una rete *wireless* altrui o leggerne i dati. Inoltre fa sempre parte del protocollo il supporto per creare e gestire una rete locale senza fili (quella che si suole chiamare una *Wireless LAN* o WLAN) in cui le macchine comunicano fra loro utilizzando un opportuno *punto di accesso*.

Nell'uso effettivo una rete *Wi-Fi* presenta poi anche delle altre differenze più sottili rispetto ad una Ethernet ordinaria, come ad esempio il fatto che, non essendovi una connessione diretta fra tutte le macchine che ne fanno parte, si possono avere situazioni in cui una macchina ne vede altre due, ma queste non si vedono fra di loro. Inoltre la tecnologia supporta diverse modalità operative, compresa la possibilità di connessioni dirette fra macchine senza dover creare una vera rete locale.

Per tutti questi motivi quando si ha a disposizione una scheda di rete *Wi-Fi* la semplice assegnazione di un indirizzo, come si farebbe per una scheda Ethernet non è sufficiente. In genere infatti una rete WLAN è gestita grazie all'uso di uno o più dispositivi che ne curano l'assemblaggio, detti *access point* (o in breve AP) dato che appunto svolgono il ruolo di *punti di accesso* alla rete stessa.

In questo caso si tratta di apparati specifici, spesso dotati anche di connessione Ethernet, il cui ruolo può essere sia quello di fare da *ponte* fra la rete *Wi-Fi* e quella Ethernet,[42] sia

[42]proprio nel senso di un *bridge* analogo a quello con cui si possono unire due tratti di rete Ethernet.

quello di fornire il supporto per la creazione di una WLAN estesa in cui le singole macchine possono collegarsi per poi comunicare fra loro (anche quando le loro interfacce non siano in grado di vedersi direttamente) con la capacità ulteriore di gestire il cosiddetto *roaming*, in cui una macchina opera sulla stessa WLAN passando da un *access point* all'altro, per poter coprire una area più vasta.

Per poter entrare nella rete, e comunicare poi con le altre macchine che ne fanno parte, è in genere necessario prima collegarsi, o meglio, per usare il gergo utilizzato dal protocollo, *associarsi*. Vedremo che questo comporta una serie di passi, fra cui è prevista la scelta della rete (trattandosi di onde radio possono sussistere reti diverse nello stesso spazio), delle modalità di collegamento e delle eventuali restrizioni di accesso.

Dato che per la natura stessa delle trasmissioni radio più *access point* possono essere raggiungibili da una stessa interfaccia, ciascuna rete *Wi-Fi* viene identificata da un opportuno identificativo denominato ESSID (*Extended Service Set ID*), una stringa alfanumerica che la distingue dalle altre presenti nello stesso spazio e che in genere viene impostata sull'*access point* (o sugli *access point*, se si costruisce una rete con più celle) che la gestiscono.

Uno dei passi che consente l'uso di una rete *Wi-Fi* è proprio quello relativo alla impostazione su una interfaccia di questo valore, che consente di scegliere a quale, fra le eventuali reti disponibili, ci si vuole *associare*. In questo caso una interfaccia di rete *Wi-Fi* deve essere configurata per operare nel cosiddetto *infrastructure mode*, cioè in una modalità operativa in cui essa diventa parte di una WLAN. In realtà questa modalità può essere usata in due modi diversi, o come *server* che fornisce un punto di accesso (il cosiddetto *AP infrastructure mode* o *manager mode*),[43] o come macchina *client* che fa parte della rete (il cosiddetto *station infrastructure mode* o *managed mode*). Quest'ultimo che è il caso più comune, quello in cui ci si vuole collegare ad un rete messa a disposizione da altri.

Oltre a questa, che è la più comune, esistono altre modalità di uso di una rete *Wi-Fi*; ad esempio si può utilizzare l'*ad-hoc mode* per creare una connessione fra diverse schede quando non sia disponibile un *access-point*; in questo caso ciascuna macchina diventa responsabile di gestire la propria rete, e non esiste un *access point* che la governa. Una ulteriore modalità di utilizzo di una scheda *Wi-Fi* è il cosiddetto *monitor mode*, usato per leggere in maniera passiva (cioè senza trasmettere nulla) i pacchetti che vengono ricevuti, e che in genere viene utilizzato per analizzare il traffico.

7.5.5 I *wireless tool* di Linux

Uno dei problemi maggiori nella gestione delle interfacce *wireless* è quello della loro estrema varietà e della complessità dei meccanismi necessari al loro funzionamento. Sono state infatti realizzate diverse varianti dello stesso standard IEEE 802.11, anche incompatibili fra di loro per frequenze radio o modalità operative. Inoltre sono disponibili diversi meccanismi di autenticazione e cifratura, anche perché quelli introdotti con la prima versione dello standard si sono dimostrati del tutto inefficaci e facilmente vulnerabili.

Questo ha comportato anche una discreta confusione anche nelle diverse realizzazioni del supporto fornito dal kernel per queste tecnologie, con implementazioni concorrenti delle varie

[43] in questo caso si vuole fare da *access point*, cosa che, se la propria interfaccia di rete supporta questa modalità, è sempre possibile anche se occorre usare del software aggiuntivo, `hostapd`, che fornisca i servizi necessari alla gestione di una WLAN.

parti dei protocolli a seconda del tipo di scheda, ed un proliferare di soluzioni ad hoc, a cui si aggiungono le problematiche dovute al fatto che non tutte le schede sono in grado di supportare tutte le modalità operative in maniera nativa. Da alcuni anni è comunque in corso una ristrutturazione completa,[44] con la creazione di una infrastruttura univoca per tutto quello che non riguarda la gestione fisica del dispositivo.

Benché nella ristrutturazione del supporto per le reti *Wi-Fi* sia in corso una anche transizione verso nuovi strumenti di amministrazione,[45] la loro diffusione comunque è ancora limitata, per cui tratteremo solo l'interfaccia classica delle cosiddette *Wireless Extension* e dei relativi *wireless-tools*, che resta tutt'ora supportata da tutte le schede anche se non viene più ulteriormente sviluppata.

Il primo comando fornito dai *wireless-tools* è `iwconfig`, che consente di effettuare le impostazioni di una interfaccia di rete per quanto riguarda i parametri specifici di una rete *wireless*. Il comando ha una sintassi simile a quella `ifconfig` e richiede come primo argomento il nome di una interfaccia di rete. Se invocato senza altri argomenti stampa le caratteristiche di quella interfaccia o la stringa *no wireless extension* se questa non è una interfaccia di rete *wireless*. Si può invocare il comando anche senza nessun argomento nel qual caso stamperà le caratteristiche di tutte le interfacce di rete presenti. Un esempio tipico di questo comando è:

```
anarres:~# iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

eth1      IEEE 802.11b  ESSID:"Laneed"
          Mode:Managed  Frequency:2.412 GHz  Access Point: 00:90:FE:72:4D:87
          Bit Rate:11 Mb/s   Sensitivity:1/0
          Retry limit:8   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality=40/70  Signal level=-62 dBm  Noise level=-98 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:239715
          Tx excessive retries:202  Invalid misc:0   Missed beacon:0
```

in cui si può notare come le prime due interfacce di rete (il *loopback* ed una scheda Ethernet ordinaria) non hanno supporto *wireless*, mentre la seconda scheda ha il supporto per la versione *IEEE 802.11b* del protocollo.

Il comando, oltre alla versione del protocollo, stampa il valore dell'ESSID a cui la scheda è associata, a questo seguono altre informazioni, come il modo (nel caso specifico *Managed*, si ricordi quanto appena detto in sez. 7.5.4) la frequenza del canale usato ed il *MAC Address* dell'*access point* e tutta una serie di ulteriori informazioni, fra cui è sempre opportuno tenere sotto controllo quelle relative alla qualità del segnale, riportate sulla sesta riga, dato che se questa cala eccessivamente si potrà perdere la connessione, o avere grossi problemi di traffico.

[44] l'evoluzione del supporto del kernel per le tecnologie *Wi-Fi* è molto complesso e non staremo a percorrerlo, basti sapere che al momento queste sono state suddivise in tre sezioni distinte, la nuova interfaccia di configurazione delle schede interna al kernel, denominata *cfg80211*, la nuova interfaccia di configurazione verso le applicazioni in *user-space*, denominata *nl80211*, e l'interfaccia per la delegazione di alcune funzionalità di gestione a programmi in *user-space*, denominata *mac80211*.

[45] questi sono realizzati attraverso un nuovo comando generico, `iw`, che consente di configurare i vari aspetti tramite altrettanti sotto-comandi in analogia al comando `ip` di sez. 7.3.4.

Parametro	Significato
essid	imposta il valore dell'ESSID, prende una stringa o il valore speciale any che su alcune schede consente di utilizzare qualunque ESSID (il cosiddetto ESSID promiscuo).
mode	imposta la modalità operativa dell'interfaccia, i valori possibili sono: Managed, nodo connesso ad una rete di *access point*; Ad-Hoc, rete senza *access point* e con una sola cella; Master nodo che agisce come *access point*; Repeater, nodo che agisce come ripetitore di pacchetti; Secondary, nodo che agisce come ripetitore o *access point* di backup; Monitor, nodo non associato a nessuna rete che ascolta passivamente.
rate	per le schede che supportano diverse velocità di trasmissione imposta quella da usare (espressa in forma bit al secondo o in automatico con auto).
key	imposta le chiavi di cifratura, espresse in forma esadecimale: indicando solo la chiave questa diventerà quella attiva, altrimenti si può appendervi un indice numerico fra parentesi quadre ([N]) per memorizzarla; indicando solo l'indice si rende la relativa chiave attiva, indicando open si consente l'uso di sessioni non cifrate, mentre aggiungendo restricted si richiede l'uso di sole sessioni cifrate.
freq	imposta la frequenza di trasmissione fra quelle disponibili (alternativa a channel).
channel	imposta il canale di trasmissione fra quelli disponibili (alternativa a freq).

Tabella 7.28: Principali parametri operativi impostabili con iwconfig.

Oltre alla stampa dello stato di una interfaccia, il comando consente anche di impostare i vari parametri funzionali della stessa, in questo caso deve essere specificato come primo argomento il nome dell'interfaccia seguito dal nome del parametro e dal relativo valore come ultimo argomento. Ad esempio col parametro essid si imposta l'ESSID della rete a cui si vuole che la scheda si associ, mentre con mode si imposta la sua modalità operativa. Si sono riportati in tab. 7.28 i parametri più rilevanti, per una trattazione completa si consulti la pagina di manuale.

Oltre a iwconfig i *wireless-tools* forniscono un secondo comando, iwlist, che viene utilizzato per ottenere informazioni relative all'interfaccia ed alla rete. Il comando richiede in genere come primo argomento l'interfaccia da usare (ma se non specificato le prova tutte) seguita dalla operazione da eseguire. Ad esempio con scanning si può ottenere la lista degli *access point* e delle celle *ad-hoc* raggiungibili dall'interfaccia, con le relative informazioni, come ESSID, frequenze/canali utilizzati, qualità del segnale ecc.

Argomento	Significato
scan	esegue una scansione per la ricerca delle reti *Wi-Fi* raggiungibili e riporta le relative informazioni.
freq	stampa la lista delle frequenze di trasmissione disponibili e dei relativi canali associati.
chan	come freq.
keys	elenca le chiavi di cifratura impostate sul dispositivo.

Tabella 7.29: Principali argomenti di iwlist.

Si è riportato in tab. 7.29 un elenco delle principali richieste eseguibili con `iwlist`, ma al solito per tutti i dettagli e l'elenco completo si consulti la relativa pagina di manuale.

7.6 I comandi diagnostici

Una volta eseguita la configurazione di base della rete in genere è opportuno controllarne il funzionamento, cosa che può essere fatta utilizzando un qualunque programma che usa un servizio di rete (il modo più comune è lanciare un browser). In caso di malfunzionamento, o comunque per eseguire controlli più approfonditi, sono disponibili i comandi che tratteremo in questa sezione, che possono essere utilizzati per verificare il buon funzionamento della rete nei suoi vari aspetti.

7.6.1 Il comando `ping`

Una volta configurate le interfacce di rete in genere il primo controllo che si va ad effettuare per vedere se la rete funziona è quello di "*pingare*" (orribile inglesismo di cui non sono riuscito a trovare una traduzione) un'altra macchina. Il nome e la terminologia derivano dall'analogia con gli impulsi sonori usati dai *sonar* ma nella nostra analogia telefonica potremmo dire che questo equivale a telefonare ad un numero per sentire se dà la linea.

Il comando `ping` permette di inviare un pacchetto ICMP (abbiamo accennato a questo protocollo in sez. 7.2.1) di tipo *echo request*. Il protocollo IP prevede che la macchina che riceve un pacchetto di questo tipo debba rispondere con l'emissione di un altro pacchetto ICMP, di tipo *echo reply* che `ping` si incaricherà di ricevere, mostrando anche il tempo intercorso fra l'invio e la risposta.

Il comando si invoca in genere passando come argomento l'indirizzo IP della macchina bersaglio. Si può usare anche il nome simbolico, ma in tal caso oltre al collegamento verso la macchina bersaglio deve funzionare anche la risoluzione dei nomi, cosa che in genere richiede la raggiungibilità del server DNS,[46] e se per un qualche motivo questo non fosse raggiungibile (o fosse lui a non funzionare) non si otterrebbe nessuna risposta anche con il resto della rete a posto. Un esempio dell'uso di `ping` è il seguente:

```
piccardi@havnor:~$ ping 192.168.168.10
PING 192.168.168.20 (192.168.168.20): 56 data bytes
64 bytes from 192.168.168.20: icmp_seq=0 ttl=255 time=0.7 ms
64 bytes from 192.168.168.20: icmp_seq=1 ttl=255 time=0.3 ms
64 bytes from 192.168.168.20: icmp_seq=2 ttl=255 time=0.3 ms

--- 192.168.168.20 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.3/0.4/0.7 ms
```

Il comando invia un pacchetto al secondo; nell'esempio riceve sempre risposta, e riporta quindi il tempo che è intercorso fra l'invio e la ricezione. Quando viene fermato (nel caso dell'esempio con `C-c`) stampa anche una statistica riassuntiva. L'esempio ci mostra che la macchina con IP `192.168.168.10` è attiva ed è raggiungibile. Il comando prende una serie di opzioni, le principali

[46] in realtà si può usare anche il nome pure in assenza di DNS, ma esso deve essere scritto in `/etc/hosts` (vedi sez. 7.4.3) o risolvibile altrimenti.

delle quali sono riportate in tab. 7.30 insieme al loro significato; per un elenco completo si può fare riferimento alla pagina di manuale.

Opzione	Significato
`-c` *count*	invia solo *count* pacchetti e poi esce stampando la statistica senza bisogno di interruzione esplicita.
`-f`	invia i pacchetti alla velocità con cui tornano indietro, o 100 al secondo, stampando un "." per ogni pacchetto inviato ed un backspace per ogni pacchetto ricevuto, così da visualizzare le perdite; solo l'amministratore può usare questa opzione che carica pesantemente la rete.
`-i` *wait*	aspetta *wait* secondi invece di uno fra l'invio di un pacchetto ed il successivo.
`-n`	non effettua la risoluzione di nomi e indirizzi.
`-p` *pattern*	si possono specificare fino a 16 byte con cui riempire il *carico* del pacchetto. Il valore *pattern* deve essere specificato come numero esadecimale. Si usa questa opzione per diagnosticare eventuali problemi di corruzione dei dati.
`-q`	sopprime tutte le stampe eccetto le statistiche finali.
`-s` *size*	invia pacchetti di dimensione `size` invece dei 56 byte usati per default.

Tabella 7.30: Principali opzioni del comando `ping`.

Si tenga presente comunque che quello che si può effettuare con `ping` è solo un controllo preliminare, se non si riceve risposta il motivo può dipendere da moltissimi fattori: da un errore di configurazione, dall'aver usato un indirizzo IP sbagliato, dalla non risposta della macchina che si vuole controllare (ad esempio potrebbe essere spenta), da una mancanza di connessione sulla rete, dal fatto che uno dei *router* che devono inoltrare i pacchetti per raggiungere la destinazione può avere un malfunzionamento. Se però va tutto bene se non altro si può concludere che la rete è attiva e funzionante ed evitare di controllare se il cavo è attaccato.

7.6.2 I comandi `traceroute`, `tracepath` e `mtr`

Un secondo comando molto utile per controllare il funzionamento di un collegamento sulla rete è `traceroute`, che serve, come dice il nome, a tracciare la strada che fanno i pacchetti per arrivare ad una destinazione su Internet.

Il comando sfrutta una caratteristica del protocollo IP che prevede, nelle informazioni associate a ciascun pacchetto, un campo chiamato TTL che viene decrementato ogni volta che il pacchetto attraversa un *router* (in quello che in gergo viene chiamato un *hop*). Quando il valore si annulla il protocollo richiede che il *router* scarti il pacchetto ed invii un messaggio ICMP (di tipo *time exceeded*) al mittente, per evitare che i pacchetti persi nei cosiddetti *routing loop* continuino a circolare sulla rete, con spreco di banda.

Di default `traceroute` invia una serie di pacchetti UDP con TTL crescente a partire da 1, così da ricevere un ICMP *time exceeded* da ogni *router* attraversato per giungere a destinazione. In questo modo si può tracciare tutta la strada fatta da un pacchetto guardando gli indirizzi da cui arrivano queste risposte. Il default del comando è di inviare tre pacchetti sonda per ogni valore progressivo del TTL ed aspettare una risposta fino a 5 secondi. Un esempio del suo utilizzo è il seguente:

```
gil:~# traceroute www.debian.it
traceroute to www.debian.it (213.254.12.146), 30 hops max, 40 byte packets
 1  vl12.core1.net.playnet.it (62.48.34.7)  0.908 ms  0.852 ms  1.065 ms
 2  vl26.mix.net.playnet.it (62.48.45.255)  7.808 ms  7.800 ms  7.788 ms
 3  g-2-5.gw-border.uli.it (62.212.3.113)  7.723 ms  7.745 ms  7.734 ms
 4  f0-1.gw-mix.uli.it (62.212.3.126)  11.920 ms  11.923 ms  11.912 ms
 5  itgate.mix-it.net (217.29.66.65)  7.672 ms  7.663 ms  7.650 ms
 6  213.254.31.245 (213.254.31.245)  16.320 ms  15.637 ms  15.605 ms
 7  213.254.0.13 (213.254.0.13)  13.045 ms  12.159 ms  12.145 ms
 8  picard.linux.it (213.254.12.146)  16.342 ms  16.342 ms  16.335 ms
```

Il comando stampa la strada fatta dai pacchetti per arrivare a destinazione, riportando in ordine progressivo per numero di *hop* l'indirizzo IP (ed il nome, se risolto) di ciascun *router* attraversato, seguito dai tempi di risposta per i tre pacchetti inviati. Si tenga presente che è normale avere rotte diverse, specie in caso di linee ridondate, per cui per un certo *hop* si possono ricevere risposte da *router* diversi, che verranno elencati sulla stessa linea.

Il comando risulta particolarmente utile quando si è appurato che la propria rete funziona almeno a livello di collegamento locale, ma non si riesce a raggiungere un qualche indirizzo su Internet. Esso infatti consente di verificare se il problema può essere imputato al fatto che la strada che prendono i pacchetti in uscita è interrotta o se si sono formati dei *routing loop*, cioè situazioni in cui, per un qualche errore di configurazione, i pacchetti vengono reinviati in cerchio attraverso una serie di *router* senza poter mai arrivare a destinazione; in tal caso infatti si osserverà il ripetersi della presenza di questi *router* nell'output di `traceroute`.

Si tenga presente però che il fatto che non si riceva risposta dalla destinazione finale non comporta necessariamente che questa sia irraggiungibile, è infatti un caso comune avere dei firewall lungo il percorso che possono bloccare o scartare i pacchetti UDP inviati dal comando. In tal caso, quando per un certo *hop* non viene ricevuta una risposta, il comando stampa una serie di asterischi. Il comando inoltre si ferma soltanto quando riceve una risposta dalla destinazione finale, che può riconoscere perché in tal caso, proprio per distinguerlo dalla scadenza durante il percorso, il codice del pacchetto ICMP *time exceeded* è diverso. Se anche la destinazione finale non dovesse rispondere, il comando prosegue a cercare la strada fino ad un numero massimo di 30 *hop*, un default che può essere cambiato con l'opzione `-m`.

Per cercare di superare questo tipo di difficoltà è possibile richiedere a `traceroute` l'uso di pacchetti di tipo diverso, ad esempio con l'opzione `-I` si può richiedere l'uso di pacchetti ICMP (come quelli di `ping`) mentre per le versioni più recenti si può invocare il comando come `tcptraceroute` o con l'opzione `-T` per fargli eseguire le richieste con pacchetti TCP.

Il comando richiede sempre come argomento l'indirizzo (numerico o simbolico) di cui si vuole tracciare la rotta, ed opzionalmente la dimensione dei pacchetti da usare; oltre a quelle citate supporta altre numerose opzioni, che permettono di impostare varie caratteristiche dei pacchetti sonda inviati e delle modalità di risposta. Al solito si sono riportate le più significative in tab. 7.31; per le restanti si faccia riferimento alla pagina di manuale.

Una alternativa a `traceroute` è `tracepath`, che effettua lo stesso tipo di controllo, anche se con un numero molto inferiore di opzioni. Il comando di nuovo richiede come argomento

[47] il valore viene usato come numero di porta di destinazione solo per TCP ed UDP mentre per ICMP indica il numero di sequenza; per UDP (il default) viene usata normalmente la porta 33434 per il primo *hop* ed il numero viene incrementato di uno per ogni *hop* successivo, mentre per il TCP viene usata la porta 80, i dettagli sulle modalità di controllo avanzate dei pacchetti sonda inviati si trovano sulla pagina di manuale.

Opzione	Significato
`-i iface`	invia i pacchetti con l'indirizzo dell'interfaccia `iface`; ha senso solo quando ci sono più indirizzi sulla stessa macchina.
`-I`	invia pacchetti ICMP invece che UDP.
`-l`	stampa anche il TTL dei pacchetti ricevuti come risposta, utile per verificare la presenza di un *routing* asimmetrico.
`-m N`	imposta a `N` il valore massimo del TTL usato (il default è di 30 hop).
`-n`	non effettua la risoluzioni di nomi e indirizzi.
`-p port`	specifica una diversa porta iniziale (al posto del default di 33434).[47]
`-s source`	usa l'indirizzo `source` come indirizzo sorgente dei pacchetti inviati.
`-T`	invia pacchetti TCP invece che UDP.
`-w time`	imposta il numero di secondi `time` da attendere per ricevere una risposta.

Tabella 7.31: Opzioni principali del comando `traceroute`.

l'indirizzo IP (o il nome) della macchina di destinazione e traccia la strada dei pacchetti. Si può usare l'opzione `-n` per impedire la risoluzione inversa degli indirizzi IP e si può specificare come secondo argomento la porta da utilizzare per i pacchetti sonda; per tutti i dettagli sul funzionamento si consulti al solito la pagina di manuale.

A differenza di `traceroute`, `tracepath` consente di utilizzare soltanto pacchetti sonda UDP, ma oltre all'indirizzo dei *router* attraversati ed ai tempi di risposta, stampa anche le eventuali variazioni di MTU (*Maximum Transfer Unit*) trovate sulla rotta. Un esempio del suo utilizzo è il seguente:

```
gil:~# tracepath www.debian.it
 1:  gil.truelite.it (62.48.34.24)                          0.185ms pmtu 1500
 1:  vl12-gw.sw100.net.playnet.it (62.48.34.1)              2.721ms
 1:  vl12-gw.sw100.net.playnet.it (62.48.34.1)              2.184ms
 2:  po2-24.asr2.playnet.it (62.48.45.102)                  0.722ms
 3:  vl27.mix.net.playnet.it (62.48.45.251)                10.717ms
 4:  g-2-5.gw-border.uli.it (62.212.3.113)                 10.747ms
 5:  f0-1.gw-mix.uli.it (62.212.3.126)                      9.968ms
 6:  itgate.mix-it.net (217.29.66.65)                      11.172ms
 7:  po1.Rumba-Monster.edge.TRN.itgate.net (213.254.0.19)  13.707ms asymm  8
 8:  213.254.12.157 (213.254.12.157)                       13.953ms
 9:  picard.linux.it (213.254.12.146)                      12.446ms reached
     Resume: pmtu 1500 hops 9 back 57
```

La conoscenza di questo valore può essere molto utile per diagnosticare problemi sulla rete, infatti, come accennato in sez. 7.3.1, ogni tecnologia di rete supporta una dimensione massima dei pacchetti che possono attraversarla, e la sua conoscenza è di fondamentale importanza nel *routing* dei pacchetti perché questi non possono essere trasmessi se eccedono questa dimensione.[48]

Se per un qualche problema l'impostazione dei valori della MTU da usare per arrivare a destinazione (la cosiddetta *Path MTU*) non viene rilevata correttamente, ci si può trovare di

[48] per questo motivo il protocollo TCP/IP prevede opportune contromisure e meccanismi per determinare automaticamente il valore della *Path MTU* di una connessione, per maggiori dettagli si consultino le sezioni 3.2.3 e 3.4.4 di [SGL] e 12.3.5 di [GaPiL].

fronte a malfunzionamenti della rete difficilmente comprensibili in cui questa pare funzionare a tratti (cioè fintanto che non si inviano o richiedono pacchetti di dimensione superiore alla *Path MTU*) per poi bloccarsi in maniera apparentemente casuale.

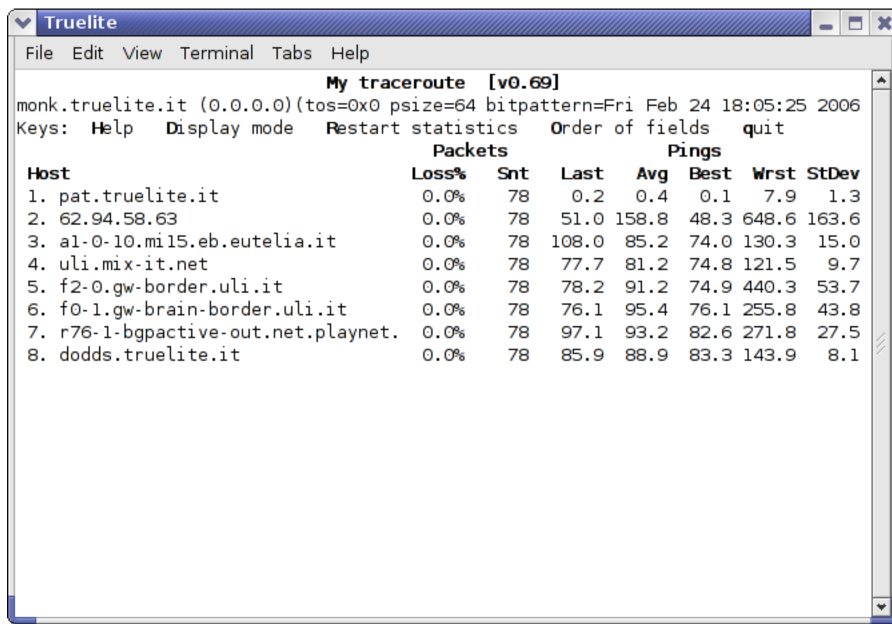


Figura 7.6: Schermata dei risultati del comando `mtr`.

Esiste infine un terzo programma, `mtr`, analogo di `traceroute` e `tracepath` ma disponibile anche in versione con interfaccia grafica, che consente di mostrare dinamicamente lo stato della strada percorsa dai pacchetti, fornendo allo stesso tempo una serie di statistiche relative al loro inoltro, unificando in sostanza le funzionalità dei due comandi e di `ping`.

Il comando prende come argomento l'indirizzo numerico o simbolico della macchina da tracciare, ed invia una serie continua di pacchetti ICMP di tipo *echo request* con TTL crescente, così può misurare sia il tempo di percorrenza che la strada seguita. Inoltre il comando, a meno di non averne preimpostato un numero specifico con l'opzione `-c`, esegue indefinitamente la ripetizione dell'invio dei pacchetti, raccogliendo e stampando in tempo reale una serie di statistiche relative alla rotta da esso tracciata. In questo caso si potrà terminare il programma con la solita interruzione (con `C-c`) o premendo il tasto "`q`".

Un esempio dell'uscita del programma, invocato in modalità testo con l'opzione `-t`, è riportato in fig. 7.6, le colonne mostrano le macchine attraversate (la cui risoluzione avviene in tempo reale), la percentuale di perdita dei pacchetti su ciascun tratto, e le statistiche sui tempi di trasmissione rilevati. Le opzioni principali di `mtr` sono riportate in tab. 7.32, per l'elenco completo si consulti la pagina di manuale.

Il grande vantaggio di `mtr` è che fornisce un monitoraggio continuo dello stato della linea, e consente così, in particolar modo grazie alla presenza dell'ordinamento dei tempi di attraversa-

Opzione	**Significato**
`-a` *`addr`*	usa l'indirizzo *`addr`* per forzare l'invio dei pacchetti in uscita sulla relativa interfaccia.
`-c` *`count`*	ripete *`count`* cicli di invio e poi termina automaticamente (senza `-r` però la schermata con i risultati viene cancellata alla terminazione del programma).
`-i` *`time`*	imposta un numero di secondi `time` da far passare fra un invio ed il successivo (il default è 1).
`-n`	non effettua la risoluzioni di nomi e indirizzi.
`-r`	abbinato a `-c` stampa la statistica finale dei risultati ottenuti alla fine dei cicli.
`-t`	forza l'uso dell'interfaccia testuale.
`-u`	invia pacchetti UDP invece che ICMP.

Tabella 7.32: Opzioni principali del comando `mtr`.

mento dei vari *router* e alla statistica dei pacchetti perduti, di identificare quali sono i tratti più problematici del traffico verso una certa macchina.

7.6.3 Il comando `netstat`

Un altro utile comando diagnostico, che permette di visualizzare una grande quantità di informazioni relative alla rete, è `netstat`. Il comando è piuttosto complesso dato che permette di ottenere informazioni riguardo a tutte le funzionalità del sistema concernenti la rete, anche se lo scopo per cui viene usato più spesso è quello di visualizzare tutte le connessioni attive su una macchina.

Il comando prevede che la prima opzione indichi il tipo di informazione da mostrare, l'elenco di quelle disponibili è riportato in tab. 7.33, ma se non se ne indica nessuna viene assunto il comportamento di default che è quello di mostrare la lista di tutti i socket aperti nel sistema.

Opzione	**Significato**
`-g`	mostra le informazioni relative ai gruppi di *multicast* di cui si fa parte.
`-i`	mostra le informazioni relative alle interfacce di rete, equivalente all'uso di `ifconfig` senza argomenti.
`-M`	mostra le informazioni relative alle connessioni mascherate dal firewall (funziona solo con i kernel della serie 2.2.x).
`-r`	mostra le informazioni relative alla tabella di instradamento (equivalente all'uso di `route` senza argomenti)
`-s`	mostra delle statistiche sommarie sull'uso dei vari protocolli di rete.

Tabella 7.33: Opzioni del comando `netstat` per il controllo del tipo di informazioni da visualizzare.

Quando viene usato con le opzioni di default il comando mostra le informazioni riguardo a *tutti* i socket aperti. La lista comprende anche i socket locali, che sono usati da vari programmi come meccanismo di intercomunicazione, e che non hanno nulla a che fare con la rete. Per questo motivo è opportuno specificare le opzioni `-t` per richiedere di visualizzare solo i socket TCP o `-u` per vedere quelli UDP, che sono quelli che riguardano le connessioni con la rete esterna. Un possibile esempio del risultato di `netstat` è allora il seguente:

```
root@gont:~# netstat -t
tcp        0      0 ppp-42-241-98-62.:32798 serverone.firenze:imaps ESTABLISHED
```

che mostra una connessione attiva verso un server di posta.

Invocato in questa maniera il comando riporta solo i dati relativi ai socket attivi, in realtà oltre a questi saranno presenti anche i socket per i quali non ci sono connessioni in corso, ma che sono in ascolto, in attesa di riceverne. Questi vengono mostrati soltanto se si usa l'opzione `-a`; nel qual caso il precedente risultato sarebbe stato:

```
root@gont:~# netstat -at
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:printer               *:*                     LISTEN
tcp        0      0 *:5865                  *:*                     LISTEN
tcp        0      0 *:webcache              *:*                     LISTEN
tcp        0      0 *:tproxy                *:*                     LISTEN
tcp        0      0 gont.earthsea.ea:domain *:*                     LISTEN
tcp        0      0 localhost:domain        *:*                     LISTEN
tcp        0      0 *:ssh                   *:*                     LISTEN
tcp        0      0 *:ipp                   *:*                     LISTEN
tcp        0      0 *:nntp                  *:*                     LISTEN
tcp        0      0 *:smtp                  *:*                     LISTEN
tcp        0      0 ppp-42-241-98-62.:32798 serverone.firenze:imaps ESTABLISHED
```

Infine usando l'opzione `-n` i risultati verranno stampati con tutti gli indirizzi ed i numeri delle porte espressi in forma numerica, senza che ne sia effettuata la risoluzione (quella di cui abbiamo parlato in sez. 7.4) nei rispettivi nomi simbolici. Le altre principali opzioni di controllo, non rientranti in quelle già illustrate in tab. 7.33, sono riportate in tab. 7.34.

Opzione	**Significato**
`-a`	mostra i dati relativi sia ai socket attivi che a quelli in ascolto.
`-c`	ripete continuamente il comando ogni secondo.
`-e`	stampa informazioni addizionali, può essere ripetuto due volte per avere ancora più informazioni.
`-l`	mostra i dati relativi ai socket in ascolto (di default non vengono mostrati).
`-n`	non esegue la risoluzione di indirizzi e porte.
`-t`	mostra i dati relativi ai socket TCP.
`-u`	mostra i dati relativi ai socket UDP.

Tabella 7.34: Opzioni principali del comando `netstat`.

Quando si usa `netstat` per verificare lo stato delle connessioni di rete, il comando genera una tabella con una voce per ciascuna di esse, come illustrato negli esempi precedenti. Il campo Proto riporta il protocollo della connessione. I campi Local Address e Foreign Address indicano gli indirizzi locale e remoto della stessa, che nel caso di socket su Internet sono nella forma:

```
indirizzo:porta
```

dove un asterisco indica un indirizzo o una porta qualunque. Il campo State indica lo stato della connessione. Una spiegazione dettagliata del significato dei valori di questo campo va di

nuovo al di là delle finalità di questo testo, e richiede una trattazione approfondita del protocollo TCP/IP,[49] per cui ci limiteremo ad alcune note panoramiche.

Delle varie righe quelle che meritano attenzione sono quelle relative agli stati `LISTEN` ed `ESTABLISHED`. Lo stato `LISTEN` indica la presenza di un programma in ascolto sulla vostra macchina in attesa di connessione, nell'esempio precedente ce ne sono vari, corrispondenti a servizi come la posta, le news, il DNS, la stampa via rete. Gli indirizzi di norma non sono specificati in quanto la connessione può essere effettuata su uno qualunque degli indirizzi disponibili sulle interfacce locali, e a partire da un qualunque indirizzo esterno. Lo stato `ESTABLISHED` indica le connessioni stabilite ed attive, e riporta nei campi degli indirizzi i numeri di IP e porta dei due capi della connessione. Altri stati che possono essere riportati dal comando sono `FIN_WAIT`, `TIME_WAIT`, che si riferiscono a connessioni che si stanno chiudendo.

Si noti anche che quando si esegue `netstat` con l'opzione `-u` per rilevare lo stato dei socket UDP, nella tabella risultante il campo `State` è vuoto anche quando ci sono servizi in ascolto, in quanto lo stato è definito soltanto per i socket TCP.

7.6.4 Il protocollo ARP ed il comando `arp`

Come accennato in sez. 7.2.1 il protocollo ARP viene usato (dal kernel) per associare ad un indirizzo fisico presente sulla rete locale, quello che per gran parte delle reti è il cosiddetto *MAC address*, al corrispondente indirizzo IP, cosicché il kernel possa sapere a quale scheda (Ethernet nel caso più comune) mandare i pacchetti.

In realtà il protocollo viene utilizzato tutte le volte che il kernel deve inviare un pacchetto IP verso l'esterno su una rete di tipo Ethernet (o con funzionalità equivalenti come il token ring o l'FDDI). Le possibilità sono sempre due, o si deve comunicare con un indirizzo IP nella stessa sottorete, nel qual caso si invierà il pacchetto direttamente al destinatario tramite l'indirizzo fisico della sua scheda di rete, o si deve inviare il pacchetto altrove passando attraverso un *gateway*, nel qual caso dalla tabella di *routing* si otterrà l'indirizzo IP di quest'ultimo, che sarà usato per ricavare il *MAC address* della scheda a cui inviare il pacchetto.

Il protocollo viene usato per mandare delle richieste in *broadcast* che vengono ricevute da tutte le schede su una stessa LAN. Queste hanno la tipica forma "*devi dire chi è `X.X.X.X` a `Y.Y.Y.Y`*" dove `X.X.X.X` è l'indirizzo IP che si vuole risolvere, ed `Y.Y.Y.Y` è quello del richiedente. Quest'ultimo viene automaticamente identificato dato che il suo *MAC address* è riportato come indirizzo sorgente nel pacchetto di richiesta, così che la macchina la cui interfaccia ha l'IP `X.X.X.X` può rispondere direttamente con un messaggio del tipo "*`X.X.X.X` è `NN:NN:NN:NN:NN:NN`*" trasmettendo il suo *MAC address* direttamente al richiedente.

In questo modo una macchina può interrogare le sue vicine sulla stessa LAN e costruire un elenco di corrispondenze. Per evitare di oberare la rete di richieste le corrispondenze trovate vengono mantenute per un certo tempo in quella che viene chiamata la *ARP cache* del kernel, e rinnovate solo dopo che sono scadute. In generale non è necessario nessun intervento di configurazione diretto per la gestione di questo protocollo, ma esistono comunque esigenze per cui può risultare utile esaminare ed eventualmente modificare manualmente la *ARP cache*.[50]

[49] una trattazione completa è in [TCPIll1], una sintesi si può trovare nelle appendici di [GaPiL].

[50] in particolare è ben noto un attacco che consiste di inserire corrispondenze arbitrarie nella *ARP cache* con pacchetti ARP "*falsi*", la tecnica si chiama *ARP poisoning* ed è trattata con maggiori dettagli nella sezione 5.4 di [SGL], se si sospetta questo tipo di attacco esaminare la *ARP cache* può risultare utile.

Ad esempio una tecnica che impiega questo protocollo è quella del cosiddetto *proxy ARP*, usata quando si divide in due reti separate una rete che prima era unica, introducendo un *router* a separare i due nuovi rami. Dato che un *router* non trasmette i pacchetti a livello di collegamento fisico, impostando su di esso un *proxy ARP* si fa sì che alle richieste ARP per gli IP posti sul nuovo tratto di rete al di là del *router* venga comunque risposto, fornendo l'indirizzo fisico della scheda del *router*. In questo modo, anche se non si è impostato su tutte le macchine una opportuna rotta statica per la nuova configurazione della rete, il *router* riceverà i pacchetti destinati al nuovo tratto, e potrà inoltrarli.

Il comando che permette di esaminare e modificare la *ARP cache* del kernel è `arp`.[51] Se chiamato senza opzioni il comando mostra il contenuto corrente della cache effettuando la risoluzione inversa degli indirizzi IP, un esempio del suo risultato potrebbe essere il seguente:

```
root@gont:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
oppish.earthsea.ea       ether   00:48:54:3A:9A:20   C                     eth0
havnor.earthsea.ea       ether   00:48:54:6A:4E:FB   C                     eth0
roke.earthsea.ea                 (incomplete)                              eth0
```

in cui per ciascuna riga è riportata la corrispondenza fra l'indirizzo IP di una macchina (o il nome, se risolto) e l'indirizzo fisico, con ulteriori informazioni come l'interfaccia di rete da cui essa è raggiungibile, oppure, in caso di richiesta senza risposta come per la terza voce, la dicitura `incomplete`.

Le due operazioni fondamentali del comando sono la cancellazione (con l'opzione `-d`) e l'inserimento (con l'opzione `-s`) di una voce nella *ARP cache*, nel caso della cancellazione occorre specificare l'indirizzo IP (o il nome) della macchina da rimuovere, con qualcosa come:

```
arp -d oppish
```

mentre se si vuole inserire una voce in più si dovranno specificare almeno indirizzo e *MAC address*, ad esempio con un comando come:

```
arp -s lorbaner 00:48:54:AA:9A:20
```

Si tenga presente però che una voce inserita in questo modo nella *ARP cache*, a meno che non si passi al comando come ulteriore argomento la direttiva `temp`, diventa "*statica*" e non scade più.Questo significa che per quella voce il protocollo ARP non verrà più usato e sarà sempre utilizzata l'associazione statica che si è impostata.

La conseguenza è che la macchina su cui si è effettuata tale operazione non sarà più in grado di accorgersi di eventuali variazioni del *MAC address* di altre macchine, ad esempio dovute ad un cambio di scheda di rete, che a quel punto diventeranno completamente irraggiungibili, indipendentemente dalle impostazioni che si potranno effettuare su quest'ultime al riguardo di interfacce ed indirizzi. Dato che questo tipo di problemi sono assai difficili da diagnosticare, è in genere sconsigliato effettuare una impostazione di voci statiche.

A parte le due già citate, l'opzione più comune per il comando `arp` è `-n`, che disabilita la risoluzione degli indirizzi. Le altre opzioni principali sono riportate in tab. 7.35, ed al solito per

[51] con il nuovo comando `ip` si può usare anche `ip neigh`, che non abbiamo trattato in sez. 7.3.4, ma il cui uso è analogo agli altri comandi della famiglia, per la sintassi si rimanda comunque alla pagina di manuale, accessibile con `man ip-neighbour`.

Opzione	Significato
`-d` *host*	cancella la voce relativa al nodo *host*.
`-f` *file*	legge i valori da immettere nella cache dal file *file*.
`-i` *iface*	stampa le voci corrispondenti alla sola interfaccia `iface`.
`-n`	non effettua la risoluzioni di nomi e indirizzi.
`-s` *host hwadd*	inserisce una voce nella cache, associando al nodo *host* l'indirizzo hardware *hwadd*.

Tabella 7.35: Principali opzioni del comando `arp`.

l'elenco completo ed i dettagli di funzionamento si può fare ricorso alla pagina di manuale del comando, accessibile con `man arp`.

Si tenga presente inoltre che come per la risoluzione dei nomi in indirizzi IP (vedi sez. 7.4.4) è possibile impostare delle voci statiche per la *ARP cache* tramite un opportuno file di configurazione. Il file è `/etc/ethers`, che contiene delle corrispondenze nella forma:

```
MAC-address indirizzo-IP
```

ed al posto dell'indirizzo IP si può anche specificare un nome simbolico, posto che questo sia risolvibile. Il formato dei *MAC address* è quello solito di sei byte esadecimali separati dal carattere "`:`" con cui vengono mostrati anche nell'uscita di `ifconfig`; un possibile contenuto di `/etc/ethers` sarà cioè qualcosa del tipo:

/etc/ethers
```
08:00:20:00:61:CA  129.168.1.245
```

7.6.5 I servizi RPC

In sez. 7.2.4 abbiamo visto come normalmente i servizi di rete vengano forniti utilizzando una porta, il cui valore li identifica univocamente. Con il crescere del numero di servizi che potevano essere resi disponibili questa assegnazione statica ha iniziato ad essere percepita come problematica, in quanto le porte possibili sono solo 65535, numero che si potrebbe esaurire con una certa facilità se se ne dovesse assegnarne una ad ogni servizio possibile.

A prima vista il problema può sembrare poco realistico, visto il numero non poi così ampio dei servizi di rete classici, ma quando si iniziò a pensare ad una modalità per rendere eseguibili via rete delle funzioni generiche attraverso opportuni servizi (i cosiddetti *Remote Procedure Call* o servizi RPC), il problema dell'assegnare una porta statica ad ogni funzione che ci si poteva inventare divenne significativo, per cui si dovette progettare anche un meccanismo che ne consentisse l'allocazione dinamica.

Il concetto che sta dietro le *Remote Procedure Call* è quello di un meccanismo di intercomunicazione generico basato sui protocolli di trasporto (sia TCP che UDP), che permetta ad un processo che gira su una certa macchina di accedere a dei servizi (nella forma di chiamate a funzioni esterne) in maniera trasparente rispetto alla rete. In sostanza una sorta di libreria eseguita via rete, in cui si demanda l'elaborazione dei dati ad una macchina remota, che restituirà i risultati attraverso la rete.

Ciascun servizio RPC è fornito da un apposito demone, in ascolto su una porta non definita a priori, in modo da poterne definire quanti se ne vuole, ma di impiegare delle porte solo per

quelli effettivamente utilizzati. Un programma che voglia utilizzare un servizio RPC deve prima rivolgersi ad un servizio speciale, il *portmapper*, che registra i servizi attivi ed è in grado di indicare la porta su cui sono forniti.

Il servizio del *portmapper* è l'unico a cui viene assegnata una porta fissa, la 111, che corrisponde al nome `sunrpc`. I servizi RPC infatti sono stati introdotti per la prima volta da Sun per i suoi sistemi Unix, da cui questo nome per il servizio associato al *portmapper*. Tutti gli altri servizi RPC quando si avviano si registrano presso il *portmapper* utilizzando un numero identificativo interno, ed indicano la porta su cui sono in ascolto; in sez. 8.4.1 vedremo che NFS utilizza questa infrastruttura.

In sostanza nel caso di servizi RPC si fa effettuare al *portmapper* una risoluzione dinamica fra un servizio che è stato registrato dal rispettivo server ed il numero di porta che questo sta utilizzando. In questo modo è possibile richiedere il servizio usando il numero identificativo, che è completamente indipendente dal numero di porta e non soggetto ai limiti di quest'ultimo.

La lista dei servizi RPC viene mantenuta in un apposito file `/etc/rpc` che è l'analogo di `/etc/services`, ed in realtà anche in questo caso si può usare il *Name Service Switch* per utilizzare altri supporti, anche se in pratica non viene mai fatto. Il formato del file prevede tre campi separati da spazi: il primo indica il nome ufficiale del servizio RPC, il secondo il numero identificativo ad esso associato, il terzo la lista, separata da spazi, degli alias del nome ufficiale. Un esempio di questo file, preso da una Debian Squeeze, è:

/etc/rpc
```
# This file contains user readable names that can be used in place of rpc
# program numbers.

portmapper      100000  portmap sunrpc
rstatd          100001  rstat rstat_svc rup perfmeter
rusersd         100002  rusers
nfs             100003  nfsprog
ypserv          100004  ypprog
mountd          100005  mount showmount
ypbind          100007
walld           100008  rwall shutdown
yppasswdd       100009  yppasswd
etherstatd      100010  etherstat
rquotad         100011  rquotaprog quota rquota
...
```

Per poter usufruire dei servizi RPC è pertanto necessario attivare il servizio *portmapper* e questo viene usualmente fatto direttamente dagli script di avvio attraverso l'uso dal demone `portmap`. Dato che questo demone è essenziale al funzionamento dei servizi RPC, esso viene installato di default da quasi tutte le distribuzioni.

Per una maggiore sicurezza, se viene usato solo per servizi utilizzati localmente, sarebbe opportuno lanciarlo con l'opzione `-i` che permette di indicare un indirizzo specifico su cui porsi in ascolto, facendogli utilizzare esclusivamente il `localhost`.[52] Per le altre opzioni si faccia riferimento come al solito alla pagina di manuale.

Se il *portmapper* è attivo si può verificare il funzionamento dei servizi RPC con il comando `rpcinfo`. Questo, se invocato con l'opzione `-p`, prende come argomento l'indirizzo di un nodo

[52] e questo è il default di Debian e della gran parte delle distribuzioni.

(se non si indica nulla usa la macchina locale) e stampa una lista di tutti i servizi registrati; un esempio potrebbe essere:

```
piccardi@hain:~$ rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp    787  status
    100024    1   tcp    790  status
```

Il comando consente anche, con l'opzione `-u`, di interrogare un nodo per la presenza di uno specifico servizio (passato come secondo argomento) e di inviare richieste in *broadcast* sulla rete per rilevare la presenza di altre macchine che forniscono servizi RPC (nel qual caso occorrerà passare come argomenti il numero del servizio e quello di versione). Per maggiori dettagli e l'elenco completo delle opzioni si faccia riferimento alla pagina di manuale.

7.7 I client dei servizi di base

Una volta che se ne sia completata la configurazione e verificato il funzionamento, l'utilizzo della rete avviene attraverso l'accesso ai servizi che su di essa vengono forniti. In generale i servizi di base sono forniti tutti secondo un'architettura *client-server* in cui c'è un programma che serve a fornire il servizio ed uno che serve ad utilizzarlo. Affronteremo più avanti la configurazione lato server di alcuni di questi servizi, qui ci limiteremo a trattare alcuni fra i client.

7.7.1 I comandi `telnet` e `netcat`

Uno dei servizi di base un tempo più utilizzato sulla rete era quello del *telnet*, nato per eseguire delle connessioni con cui poter operare via rete su un terminale su una macchina remota. Dato che tutti i dati vengono trasmessi in chiaro sulla rete, password comprese, questo uso è assolutamente da evitare, ed a questo scopo oggi viene completamente sostituito dal comando `ssh`, che tratteremo in sez. 8.3.2.

In ogni caso il comando `telnet` continua ad esistere, e può essere usato allo scopo originale su una rete locale sicura o quando non esiste una versione di SSH per la macchina in questione, come accade per vari apparati di rete, che forniscono un accesso alla console di amministrazione solo in questo modo. Un suo uso più interessante però è quello diagnostico, allo scopo di verificare la accessibilità e la funzionalità dei servizi.

Il comando prende come argomento la macchina cui collegarsi ed opzionalmente la porta, specificata per numero o per nome del servizio. Non specificando nessuna porta il comando si collega sulla porta 23, che corrisponde al servizio standard *telnet*, mostrando, qualora sia attivo il relativo server, una schermata di login (o quanto previsto).

Il comando supporta varie opzioni, per lo più relative alle modalità di gestione della sessione di terminale remota. Dato che questo uso è assolutamente sconsigliabile non le riportiamo neanche, i più curiosi possono consultare la pagina di manuale accessibile con `man telnet`. Di norma si usa il comando indicando semplicemente un indirizzo ed una porta, ad esempio:

```
piccardi@oppish:~$ telnet localhost 25
Trying ::1...
```

```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 oppish.earthsea.ea ESMTP Postfix (Debian/GNU)
```

in questo caso lo si è usato per connettersi alla porta 25 della propria macchina, per verificare che il relativo servizio SMTP (cioè il server per la ricezione e la trasmissione della posta elettronica) sia attivo: si noti che si è poi anche ottenuta la risposta del server che nel caso in questione è *Postfix*.

Con la stessa sintassi si può usare il comando per verificare la presenza e l'attività dei servizi sulle relative porte. Anche se non si riceve nessuna risposta (ad esempio perché il servizio interrogato necessita di una richiesta prima di rispondere), la presenza della riga `Connected` indicherà che la connessione è riuscita ed è attiva. Se invece tentiamo di collegarci verso una porta su cui non è attivo nessun server otterremo un errore di connessione rifiutata del tipo:

```
piccardi@oppish:~$ telnet localhost 80
Trying 127.0.0.1...
telnet: Unable to connect to remote host: Connection refused
```

con un ritorno pressoché immediato alla riga di comando.

Questo è il comportamento normale, ma quando la macchina contattata è protetta da un firewall normalmente tutti i pacchetti in ingresso non diretti ai servizi consentiti vengano scartati; questo comporta che un tentativo di connessione su una porta senza servizi sia completamente ignorato. In questa evenienza, non ricevendo nessuna risposta,[53] il comando `telnet` resta bloccato in attesa fino alla scadenza del timeout (il default è circa 30 secondi), riportando poi un errore di connessione.

Un secondo comando che si può utilizzare per effettuare una verifica sui servizi è `netcat`, che può essere invocato anche in forma più compatta con `nc`. Il grande vantaggio rispetto a `telnet` è che mentre quest'ultimo funziona soltanto su TCP, `netcat` può essere anche usato su UDP, e permette quindi di controllare anche i servizi che usano questo protocollo.

Inoltre `netcat`, come suggerisce il nome stesso, si limita a leggere e scrivere i dati su delle connessioni di rete associando *standard input* e *standard output* alla connessione su cui lavora, supporta quindi la redirezione ed il *pipelining*, ed è particolarmente adatto ad essere utilizzato anche all'interno di script. Questo non è possibile con `telnet`, che associa alla connessione un terminale, ed è quindi soggetto alla *disciplina di linea* di quest'ultimo.[54] Inoltre `telnet` interpreta alcuni caratteri come caratteri di controllo (eliminandoli dal flusso dei dati) e scrive alcuni dei suoi messaggi interni sullo *standard output*.

Al contrario `nc` preserva rigorosamente il flusso dei dati, e se da una parte questo lo rende meno adatto all'amministrazione via rete (ma non lo sarebbe comunque, non essendo prevista la cifratura della connessione), dall'altra lo rende uno strumento molto flessibile per inviare e ricevere dati dalla rete, e manipolarli usando la riga di comando, tanto che la pagina di manuale lo classifica come "il *coltellino svizzero* del TCP/IP".

[53] il protocollo TCP prevede, in caso di connessione su una porta su cui non ci sono servizi, l'invio di un pacchetto di risposta al client che ha effettuato il tentativo per notificare l'errore; è grazie alla ricezione di questo pacchetto che `telnet` riporta l'errore di connessione rifiutata mostrato nell'esempio precedente, ma se il tentativo di connessione viene scartato a priori non si riceverà neanche la risposta di errore.

[54] si chiama *disciplina di linea* la modalità con cui il dispositivo di terminale gestisce l'I/O, ad esempio il fatto che l'input è bufferizzato linea per linea.

La forma più elementare di invocazione del comando è identica a quella di `telnet`, in cui si specifica il nome di una macchina (o il suo indirizzo) e la porta che si desidera contattare. Ripetendo l'esempio precedente avremo:

```
piccardi@hain:~$ nc localhost 25
220 hain.truelite.it ESMTP Postfix (Debian/GNU)
```

e come si può notare in questo caso non viene scritto niente che non sia quanto stato ricevuto dalla rete; se si vogliono avere i messaggi diagnostici occorre utilizzare l'opzione `-v`, che li stamperà rigorosamente sullo *standard error*. Inoltre ripetendo due volte l'opzione il comando diventa più prolisso. Se il servizio che si vuole contattare usa il protocollo UDP basterà utilizzare l'opzione `-u`.

Un'altra caratteristica interessante del programma è che si possono specificare più porte di destinazione, ed i dati verranno inviati e letti su tutte quante. Questo permette anche di eseguire una scansione elementare per la presenza di servizi attivi, che potrebbe essere realizzata con qualcosa del tipo:

```
echo QUIT | nc -w 1 localhost 25 1-1024 6000-7000
```

dove si è usata l'opzione `-w`, che consente di specificare un tempo massimo di durata in secondi, per chiudere le connessioni che altrimenti resterebbero aperte.

Un'altra caratteristica interessante di `netcat` è che può essere utilizzato in *listen mode* (cioè come "*server*") usando l'opzione `-l` che mette il programma in ascolto sulla porta specificata con l'opzione `-p`. In tal caso sarà possibile collegarsi da remoto al programma, che scriverà sullo *standard output* quanto ricevuto sulla connessione.

In questo modo ad esempio è possibile trasferire un file via rete redirigendo lo *standard output* del comando, utilizzando `nc` sull'altro capo della connessione per inviarlo. Infine usando l'opzione `-e` è possibile indicare un programma che verrà eseguito collegando *standard input* e *standard output* alla connessione. Questo significa che quanto si invierà sulla connessione arriverà in ingresso al programma e si riotterranno sulla stessa i dati in uscita. Così si può trasformare `nc` (se ad esempio si fa eseguire la shell) in un surrogato temporaneo del servizio *telnet*.

Opzione	**Significato**
`-e` *cmd*	esegue il file *cmd*, collegando il relativo *standard input* e *standard output* alla connessione.
`-l`	si pone in ascolto sulla porta specificata con `-p`.
`-p` *port*	indica la porta (o le porte indicando un intervallo separato da un "-") su cui mettersi in ascolto; da usare insieme a `-l`.
`-q` *secs*	attende `secs` secondi dopo la chiusura dello *standard input*.
`-s` *addr*	specifica l'indirizzo sorgente locale.
`-u`	esegue le connessioni su UDP.
`-v`	modalità *prolissa*, stampa informazioni sullo *standard error*.
`-w` *secs*	aspetta *secs* secondi per il timeout.

Tabella 7.36: Principali opzioni del comando `netcat`.

In tab. 7.36 si sono riportate le principali opzioni del comando, al solito per l'elenco completo si può fare riferimento alla pagina di manuale.

7.7.2 Il comando `ftp`

Il protocollo FTP è uno dei più vecchi protocolli che consentono lo scambio di file su Internet. Il protocollo permette di prelevare od immettere file su un server FTP, previa autenticazione analoga a quella del login o dell'accesso con telnet. Una sua forma particolare è il cosiddetto *FTP anonimo* in cui il servizio viene utilizzato per distribuire i file posti in un server; in questo caso il servizio non richiede autenticazione, e di norma consente solo il prelievo dei file.

Dato che il protocollo, come *telnet*, non prevede alcuna cifratura dei dati, è meglio non usarlo in quanto l'autenticazione verrebbe eseguita in chiaro; unica eccezione è la modalità anonima in cui non c'è autenticazione che è ancora una modalità molto usata per distribuire pubblicamente file.[55] Qualora si necessiti di un meccanismo per lo scambio di file via rete fra i vari utenti sono disponibili alternative come `sftp`, un programma che ha esattamente la stessa sintassi di `ftp`, ma consente l'uso di una connessione cifrata attraverso il protocollo SSH che tratteremo in sez. 8.3.

Oggi esistono molti client per il protocollo FTP, sia grafici che testuali, ma il comando originale usato per lanciare il client testuale è semplicemente `ftp`. Un esempio di uso generico del comando è il seguente:

```
piccardi@anarres:~$ ftp ftp.linux.it
Connected to vlad-tepes.bofh.it.
220 (vsFTPd 2.0.7)
Name (ftp.linux.it:piccardi): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

eventuali opzioni, ed anche la stessa risposta, dipendono in genere dalla versione del comando che si è installata, e vanno verificate facendo riferimento alla relativa pagina di manuale.

Quella illustrata nell'esempio è la risposta data dal comando `ftp` che si trova su Debian Squeeze, derivato dalla prima realizzazione del programma nato con BSD, alternative possono essere client più evoluti come `lftp` o `ncftp` che supportano la storia dei comandi, il completamento dei nomi, e capacità aggiuntive (come il download ricorsivo dei contenuti di una directory).

Si noti come nell'uso generico sia sufficiente indicare la macchina cui ci si vuole collegare. Nel caso ci viene notificato l'*hostname* effettivo del server e ci viene richiesto un nome di login che di default corrisponde al nostro username. Avendo contattato un FTP anonimo l'utente da usare è `anonymous`, specificato il quale ci viene richiesta una password, che nel caso è ininfluente (qualunque cosa si scriva sarà garantito l'accesso).

Fatto questo si viene portati su una riga di comando introdotta dal un prompt "`ftp>`" dalla quale sarà possibile inviare, scrivendoli come si farebbe con la shell, i vari comandi previsti dal protocollo per operare con il server. I più importanti sono stati brevemente illustrati in tab. 7.37, mentre la lista completa con tutti i dettagli è al solito disponibile nella pagina di manuale, accessibile con `man ftp`.

[55] anche questo utilizzo è comunque ampiamente discutibile, in quanto sarebbe comunque più efficiente distribuire i suddetti dati via web usando HTTP, che sovraccarica meno la rete e non presenta tutte le problematiche di funzionamento, in particolare quando si devono attraversare dei firewall, che ha FTP.

Comando	Significato
`ls`	stampa la lista dei file (sul server).
`cd`	cambia directory (sul server).
`lcd`	cambia directory (sul client).
`pwd`	stampa la directory corrente (sul server).
`ascii`	modalità di trasferimento di file di testo.
`binary`	modalità di trasferimento di file binari.
`pasv`	abilita i trasferimenti in modo passivo.
`get` *`file`*	scarica il file *`file`* dal server.
`put` *`file`*	invia il file *`file`* sul server.
`mget` *`files*`*	scarica i file che corrispondono alla wildcard *`files*`*.
`mput` *`files*`*	invia i file che corrispondono alla wildcard *`files*`*.
`open` *`host`*	apre una connessione verso il server *`host`*.
`quit`	chiude la sessione.
`rmdir`	cancella directory (deve essere vuota).
`mkdir`	crea directory.
`chmod`	modifica i permessi del file.

Tabella 7.37: Comandi del protocollo FTP.

Si tenga presente però che non tutti i comandi possono essere supportati dal server. Ad esempio il comando `pasv`, che abilita il cosiddetto *modo passivo*,[56] è eseguibile solo se il server supporta questa modalità di operazione, mentre i comandi `ascii` e `binary` non hanno nessun significato su sistemi unix-like, ma possono averlo per altri sistemi come il VMS o il DOS che introducono una distinzione fra questi tipi di file.

7.7.3 I comandi `finger` e `whois`

Il comando `finger` viene usato fin dalle origini di Unix, dove agiva solo sulla macchina locale, per riportare informazioni relative agli utenti di un sistema. In questo modo si poteva consentire agli altri utenti di sapere chi era collegato alla macchina. Il servizio è stato trasferito sulla rete, ed il client funziona sia in locale che in remoto (nel qual caso si eseguirà una richiesta con un parametro del tipo `nomehost`).

Un possibile esempio di uso del comando, fatto in locale, è il seguente:

```
piccardi@gont:~$ finger franci
Login: franci                         Name: Francesco Piccardi
Directory: /home/franci               Shell: /bin/bash
Home Phone: 0XX-XX000XX
Last login Tue Jun 17 21:22 (CEST) on :0
No mail.
No Plan.
```

e come si vede il programma mostra una serie di informazioni riguardo l'utente `franci`; questi può aggiungervi ulteriori informazioni creando un file `.plan` nella sua home directory, che verrà mostrato all'esecuzione del comando.

[56] il protocollo FTP prevede l'uso di due porte, la connessione avviene sempre da parte del client sulla porta 21 del server, ma su di essa vengono solo inviati i comandi, quando si richiede l'invio di file questo viene effettuato dal server con una seconda connessione che contatta il client sulla porta 20; dato che normalmente i firewall bloccano le connessioni entranti, il modo passivo fa sì che sia sempre il client a creare la nuova connessione anche per i dati.

Il comando prende come argomento un nome utente che viene confrontato sia con lo username che con il nome reale dello stesso memorizzato su `/etc/passwd`; qualora si voglia eliminare quest'ultimo confronto si può usare l'opzione `-m`. Se invocato senza argomenti mostra l'elenco degli utenti collegati e relativo terminale come:

```
piccardi@gont:~$ finger
Login     Name              Tty      Idle  Login Time   Office     Office Phone
piccardi  Simone Piccardi  *:0            Jul 21 21:55
```

altre due opzioni utili sono `-s` che mostra una lista semplice come questa e `-l` che mostra una lista lunga come la precedente, per i dettagli si può fare riferimento alla pagina di manuale.

Dato che il comando fornisce delle informazioni che potrebbero essere utili per attività non troppo benevole o per violare la privacy degli utenti (ad esempio l'uso dell'indirizzo di posta elettronica per inviare dello spam), oggi si tende sempre di più a non attivare questo servizio sulla rete. Talvolta però esso viene utilizzato per pubblicare delle informazioni, non legate alla presenza di specifici utenti sulla macchina, come eventuali recapiti e contatti, oppure per distribuire le chiavi GPG degli utenti (usando un file `.pgpkey` nella home directory degli stessi).

Il comando `whois` viene usato per contattare i server che forniscono il servizio *whois*. Questo servizio permette di accedere alle informazioni presenti nel database dei titolari dei domini su Internet, che vengono mantenute dagli enti che sono responsabili della registrazione degli stessi (in Italia il NIC). In genere lo si utilizza quando si cercano informazioni di natura amministrativa relativa ai domini, ad esempio per avere un indirizzo di contatto dei relativi proprietari.

Il comando prende come argomento una stringa di ricerca, normalmente un nome a dominio, e cerca di determinare automaticamente qual'è il server più opportuno a cui rivolgersi.[57], il quale restituisce le informazioni che verranno stampate a video, un esempio di uso del comando è:

```
piccardi@gont:~$ whois truelite.it
*******************************************************************
* Please note that the following result could be a subgroup of      *
* the data contained in the database.                               *
*                                                                   *
* Additional information can be visualized at:                      *
* http://www.nic.it/cgi-bin/Whois/whois.cgi                         *
*******************************************************************

Domain:             truelite.it
Status:             ACTIVE
Created:            2002-08-19 00:00:00
Last Update:        2009-04-09 00:03:35
Expire Date:        2010-03-24

Registrant
  Name:             TRUELITE SRL
  Organization:     TRUELITE SRL
  ContactID:        TRUE12-ITNIC
  Address:          VIA MONFERRATO, 6
                    Firenze
                    50142
                    FI
```

[57] nel caso non riesca a determinarne uno più specifico, viene contattato il server di riferimento internazionale, che è `whois.networksolutions.com`.

```
                 IT
Created:         2007-03-01 10:50:20
Last Update:     2007-03-01 10:50:20

...
```

e come si può notare si ottengono tutta una serie di informazioni relative a chi gestisce (come il titolare) un nome a dominio.

Si può richiedere al comando di contattare un server specifico, usando l'opzione `-h`, mentre con `-H` si indica al comando di non mostrare le note legali che molti server inviano. Per i dettagli sulle altre opzioni si consulti la pagina di manuale o si invochi il comando senza argomenti.

Capitolo 8

La gestione dei servizi di base

8.1 I programmi di ausilio alla gestione dei servizi di base

Tratteremo in questa sezione alcuni programmi di ausilio per la gestione dei servizi di rete più semplici, ed in particolare l'uso di un cosiddetto "*super-demone*" per gestire servizi usati occasionalmente, e l'uso dei *TCP wrappers* per fornire un meccanismo elementare di controllo degli accessi anche a quei programmi che non ne hanno uno al loro interno.

8.1.1 I servizi elementari e i super-demoni

Fino a qualche tempo fa nell'installazione di un sistema di tipo Unix venivano di norma attivati una serie servizi di rete elementari, usabili a scopo di test, come `echo` o `discard`, a cui si potevano aggiungere altri servizi minori, mantenuti attivi in previsione di un possibile futuro utilizzo. Ma mantenere perennemente in esecuzione dei demoni per fornire dei servizi usati solo in maniera occasionale (come potevano essere `telnet` o `ftp`), veniva a costituire, specie nelle prime versioni di Unix, uno spreco di risorse non accettabile.

Per questo motivo venne introdotto il concetto di *super-demone*, un demone speciale che facesse da intermediario mettendosi in ascolto sulle varie porte interessate, per poi smistare le richieste ai programmi che implementano il servizio richiesto. Oggi le esigenze di prestazioni non sussistono più, ed inoltre una buona norma di sicurezza prevede di non lasciare servizi attivi se questi non sono in uso, ma per fornire servizi minori l'uso di un *super-demone* resta l'opzione più utilizzata.

In sostanza un *super-demone* svolge quello che è il lavoro di un centralinista, che riceve le telefonate sul centralino e poi passa ciascuna di esse all'ufficio competente. Nel caso specifico il programma riceve le connessioni su una certa porta e poi lancia automaticamente il relativo programma di risposta. In questo modo non c'è bisogno di mantenere sempre in esecuzione tanti programmi diversi, ma li si possono lanciare solo quando servono.

Inoltre, come avviene per il centralinista che ti può dire direttamente l'orario degli uffici, un *super-demone* è in genere in grado di rispondere direttamente per alcuni servizi elementari (come `echo` o `discard`), per i quali non necessita di invocare nessun programma esterno.

Infine una caratteristica comune dei *super-demoni* è che sono in grado di associare alla connessione ad una porta l'esecuzione di un programma qualunque. È possibile cioè, in occasione di una connessione ad una porta, lanciare un programma collegando il socket della connessione allo *standard input* e allo *standard output* del programma stesso. Così ad esempio si potrà attivare il servizio di rete `netstat` sulla porta 15, usando l'omonimo programma visto in sez. 7.6.3.

Di norma i servizi più importanti, come il web, la posta, il DNS, o SSH non vengono utilizzati in questo modo, perché questa interposizione di un altro programma avrebbe effetti negativi sulle prestazioni, anche se in molti casi l'importanza del servizio (ad esempio quella di un server web o di posta elettronica) va commisurata con l'uso che se ne fa: per una macchina client tenere sempre attivo Apache per fornire delle pagine locali consultate ogni tanto può effettivamente essere eccessivo.

Tradizionalmente il programma utilizzato per svolgere questo compito è `inetd`, che fino a qualche anno fa veniva installato di default da tutte le distribuzioni; `inetd` però presenta numerosi limiti, ed oggi tende ad essere sostituito con il più recente `xinetd`, se non ad essere eliminato del tutto. In generale infatti l'utilità di un *super-demone* sta venendo meno, considerato che il costo sempre minore delle risorse ne rende meno vantaggioso l'utilizzo.

8.1.2 Il super-demone `inetd`

Come appena accennato la gestione dei servizi di base a lungo è stata effettuata tramite il programma `inetd`, che ancor oggi viene usato da alcune distribuzioni. Il programma viene lanciato dagli script di avvio, ma al solito si può avviare e fermare il demone manualmente con il relativo script, che di norma è `/etc/init.d/inetd`.

In caso di necessità si può eseguire `inetd` anche direttamente a riga di comando, nel qual caso si può usare l'opzione `-d` per farlo partire in modalità di debug, in cui non si distacca dal terminale. Il comando prende come argomento un file di configurazione alternativo, altrimenti utilizza il default. Per le altre opzioni ed i dettagli del funzionamento si faccia riferimento come al solito alla pagina di manuale.

Il file di configurazione di default di `inetd` è `/etc/inetd.conf`, ed è qui che si specifica su quali porte il demone deve porsi in attesa e quali programmi lanciare. Un esempio di questo file è il seguente:

/etc/inetd.conf
```
# /etc/inetd.conf:  see inetd(8) for further informations.
#
# Internet server configuration database
#
# <service_name> <sock_type> <proto> <flags> <user> <server_path> <args>
#
#:INTERNAL: Internal services
#echo           stream  tcp     nowait  root    internal
#echo           dgram   udp     wait    root    internal
#chargen        stream  tcp     nowait  root    internal
#chargen        dgram   udp     wait    root    internal
#discard        stream  tcp     nowait  root    internal
#discard        dgram   udp     wait    root    internal
#daytime        stream  tcp     nowait  root    internal
#daytime        dgram   udp     wait    root    internal
#time           stream  tcp     nowait  root    internal
```

```
#time           dgram   udp     wait    root    internal
#:MAIL: Mail, news and uucp services.
nntp            stream  tcp     nowait  news    /usr/sbin/tcpd  /usr/sbin/leafnode
#:INFO: Info services
ident           stream  tcp     wait    identd  /usr/sbin/identd        identd
```

come si vede il formato è relativamente semplice: una tabella in cui ogni riga è relativa ad un servizio da lanciare, ed i cui campi sono separati da spazi o tabulatori. Al solito righe vuote e inizianti con "#" sono ignorate.

Il servizio da fornire, o meglio la porta su cui porsi in ascolto, è identificato dal primo campo tramite il nome simbolico di `/etc/services`. Il successivo campo indica il tipo di socket: i due tipi più comuni sono `stream` o `dgram`, ma si possono usare tutti i tipi di socket supportati con Linux, (come `raw`, `rdm`, o `seqpacket`). Il terzo campo indica il protocollo usato: di norma si tratta di `tcp` o `udp`,[1] che corrispondono rispettivamente ai tipi `stream` e `dgram`.

Il quarto campo indica come `inetd` deve gestire la porta nell'esecuzione del programma. I casi sono due, se il server lanciato resta in esecuzione trattando direttamente eventuali connessioni multiple si deve usare `wait`, in tal caso `inetd` attenderà che il programma termini prima di rimettersi in ascolto sulla porta. Se invece il programma lanciato gestisce la singola connessione, è necessario che `inetd` si rimetta in ascolto sulla porta e lanci un'altra istanza in caso di nuovo collegamento, in tal caso occorre usare `nowait`.

In genere per i socket di tipo `dgram` si usa `wait` perché non esistendo le connessioni i programmi hanno la necessità di trattare tutti i pacchetti che arrivano sulla stessa porta, e si dovrà aspettare la fine del processo corrente, che indica la fine del flusso di dati, prima di rilanciare un'altra istanza. Se questo non è necessario, come avviene in genere per i socket di tipo `stream`, in cui si passa al programma il socket della connessione su cui esso fornisce una risposta, deve essere usato il valore `nowait`. Benché questi siano i casi più comuni, ma non esiste una regola specifica attinente al protocollo, e tutto dipende dal funzionamento del programma lanciato.

Il quinto campo indica l'utente per conto del quale viene eseguito il programma; se si vuole specificare anche un gruppo diverso dal gruppo principale dell'utente, lo si può fare scrivendo il campo nella forma `user.group`. Il sesto campo deve indicare il pathname completo del comando che verrà eseguito in caso di connessione, o la parola chiave `internal` se quel servizio è fornito direttamente da `inetd`, come accennato infatti `inetd` è in grado di fornire direttamente alcuni servizi elementari come `echo`, `discard` o `daytime`, per i quali non è necessario lanciare nessun programma a parte.

Eventuali argomenti dovranno essere specificati di seguito; dato però che `inetd` usa direttamente il resto della linea per invocare la funzione `exec`, questi non potranno essere specificati normalmente, riprendendo quanto si scriverebbe sulla shell, ma dovranno essere preceduti dall'argomento iniziale che indica il nome del programma lanciato. Sulla riga di comando infatti è la shell che costruisce automaticamente questo argomento usando il nome del programma invocato, in questo caso deve essere fatto esplicitamente.

Come si può notare nell'esempio sono abilitati solo due servizi, il primo è un server news gestito tramite il programma `leafnode`, il secondo è il servizio *ident*, definito dall'RFC 1413, che

[1] in realtà si può specificare un qualunque altro protocollo col nome riportato in `/etc/protocols`, ma benché sia possibile usare il demone con qualunque tipo di socket e protocollo di rete, in pratica sono questi due gli unici che vengono utilizzati.

permette di identificare l'utente proprietario del processo che ha creato una certa connessione. Si noti anche come nel caso di `leafnode` questo non sia stato lanciato direttamente ma invocato attraverso `tcpd`, un utile programma di sicurezza che permette di controllare gli accessi ai servizi di rete, come vedremo in sez. 8.1.5.

Se si vuole abilitare un nuovo servizio `netstat`, che fornisca via rete i dati delle connessioni presenti sulla nostra macchina utilizzando il programma omonimo trattato in sez. 7.6.3, basterà aggiungere al file una riga del tipo:

```
netstat    stream  tcp   nowait  nobody     /bin/netstat netstat -ant
```

e si noti come si sia ripetuto, prima dei parametri `-ant`, il nome del programma `netstat`. Senza questa ripetizione il risultato sarebbe stato che all'esecuzione di `/bin/netstat` il nome del processo sarebbe risultato `-ant`, mentre le opzioni sarebbero state perse.[2]

Fatta l'aggiunta citata, dopo aver riavviato `inetd` per fargli prendere le modifiche effettuate alla configurazione, si potrà verificare il funzionamento del nuovo servizio usando `telnet` per collegarsi verso di esso, con qualcosa come:

```
piccardi@gont~:$ telnet localhost netstat
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State
tcp        0      0 0.0.0.0:15             0.0.0.0:*              LISTEN
tcp        0      0 0.0.0.0:111            0.0.0.0:*              LISTEN
tcp        0      0 192.168.1.1:53         0.0.0.0:*              LISTEN
tcp        0      0 127.0.0.1:53           0.0.0.0:*              LISTEN
tcp        0      0 0.0.0.0:22             0.0.0.0:*              LISTEN
tcp        0      0 0.0.0.0:631            0.0.0.0:*              LISTEN
tcp        0      0 0.0.0.0:25             0.0.0.0:*              LISTEN
tcp        0      0 192.168.1.1:32777      195.110.124.18:993     ESTABLISHED
tcp        0      0 127.0.0.1:32944        127.0.0.1:15           ESTABLISHED
tcp        0      0 192.168.1.1:32778      62.177.1.107:5223      ESTABLISHED
tcp        0      0 192.168.1.1:32772      192.168.1.168:5901     ESTABLISHED
tcp        0      0 127.0.0.1:15           127.0.0.1:32944        ESTABLISHED
Connection closed by foreign host.
```

ottenendo il risultato del comando `netstat` eseguito sulla macchina remota in cui compare, nell'ultima riga, anche la connessione su cui lo stiamo leggendo.

Il precedente esempio funziona perché quando lancia un programma, `inetd` fa sì che *standard input*, *standard output* e *standard error* siano associati al socket su cui è stata aperta la connessione. Si può così eseguire un programma qualsiasi, e questo accetterà i dati in ingresso dal socket e su di esso scriverà le sue risposte; in generale è sempre compito del programma messo in esecuzione gestire opportunamente la connessione di rete, fornendo il servizio richiesto.

Nell'esempio appena mostrato quello che è successo è che il programma `netstat` è stato eseguito sul server remoto (il fatto che si sia usato `localhost` è stato solo per comodità) e ha prodotto la lista dei socket attivi scrivendola sullo *standard output*, per poi terminare. Questo ha fatto sì che noi la ricevessimo all'altro capo del socket creato dalla connessione effettuata con

[2] cosa che si può osservare direttamente eseguendo `ps`, posto di essere così veloci e fortunati da riuscire a lanciarlo mentre `netstat` sta ancora girando.

telnet, che è stato automaticamente chiuso alla terminazione del processo. La stessa procedura è applicabile in generale a qualunque comando di shell, per cui potremmo definire un servizio di rete ps (assegnandogli una porta in /etc/services) che ci fornisca l'elenco dei processi, ed in generale potremmo anche crearci un servizio ad hoc usando degli script.

8.1.3 Il super-demone xinetd

Il programma xinetd nasce come riscrittura di inetd per eseguire lo stesso compito: far partire i server appropriati in caso di connessione al relativo servizio, evitando di lanciare e tenere in memoria dei programmi che resterebbero dormienti per la gran parte del tempo.

Rispetto ad inetd esso supporta nativamente il controllo di accesso con i *TCP wrappers* (vedi sez. 8.1.5), ma oltre a questo ha una serie di funzionalità ulteriori come la possibilità di mettere a disposizione i servizi in orari determinati, dei meccanismi di redirezione delle connessioni, delle capacità di registrazione degli eventi più estese, dei meccanismi di protezione nei confronti delle scansioni delle porte, e la capacità di limitare il numero di istanze del server lanciate per resistere ad attacchi di *denial of service*.

Le maggiori funzionalità comportano ovviamente il prezzo di una maggiore complessità di configurazione, comunque il comando supporta una opzione, -inetd_compat, che gli permette di operare in modalità di compatibilità con inetd. In tal caso infatti il programma prima legge i suoi file di configurazione, e poi /etc/inetd.conf facendo partire i servizi definiti in quest'ultimo.

Il file principale di configurazione di xinetd è /etc/xinetd.conf, un esempio del suo contenuto che illustra le principali direttive generiche, è il seguente (preso dal default installato su una Debian Wheezy senza le righe di commento):

/etc/xinetd.conf
```
# Simple configuration file for xinetd
defaults
{
        instances   = 25
        log_type    =  SYSLOG auth
        log_on_success = HOST PID
        log_on_failure = HOST RECORD
        per_source  = 5
}
includedir /etc/xinetd.d
```

in questo caso nel file è presente soltanto la sezione speciale defaults che contiene i valori di default delle opzioni, da applicare per tutti i servizi per i quali essi non sono stati esplicitamente specificati.

La riga finale con la direttiva includedir richiede poi di includere automaticamente nella configurazione il contenuto di tutti i file contenuti nella directory specificata (di norma, come nel caso, si usa /etc/xinetd.d), cosa che permette di attivare in maniera indipendente un servizio, con la semplice installazione di un file in tale directory.

Al solito le righe vuote e il cui primo carattere è "#" vengono ignorate; per il resto per ciascun servizio che si vuole attivare è necessario specificare una voce (nel caso basterà scrivere un file che la contenga in /etc/xinetd.d) che inizia con la direttiva service; la sintassi generica di tale voce è del tipo:

```
service <nome_servizio>
{
      <attributo> <operatore> <valore> <valore> ...
      ...
}
```

dove <nome_servizio> indica il servizio che si vuole fornire, mentre gli attributi permettono di specificarne le caratteristiche, con un operatore di assegnazione che nella gran parte dei casi è "=", con significato ovvio. L'operatore però può essere anche "+=" o "-=", rispettivamente per aggiungere o togliere valori, anche se solo alcuni attributi supportano questa sintassi.

Come mostrato nell'esempio precedente, anche la direttiva `default` prende degli attributi; in genere questi sono attributi generali che possono a loro volta essere rispecificati in maniera diversa per i singoli servizi. Nell'esempio il primo attributo è `instances` che permette di porre un limite massimo al numero di istanze dello stesso programma server che il programma può lanciare, mentre il secondo attributo è `log_type`, che permette di specificare le modalità con cui viene effettuata la registrazione dei dati delle connessioni. Queste possono essere `FILE` per specificare a seguire un file su cui salvare direttamente i dati, o `SYSLOG` per specificare l'uso del *syslog* (vedi sez. 3.2.3), indicando poi con l'ulteriore parametro (nel caso `auth`) quale *facility* utilizzare; è possibile anche specificare di seguito una *priority*, se diversa dal default che è `info`.

I due attributi successivi, `log_on_success` e `log_on_failure`, permettono di specificare cosa scrivere nei log in caso rispettivamente di successo e fallimento di una connessione. Le indicazioni `HOST` e `USERID` sono comuni ad entrambi e permettono di registrare rispettivamente l'IP e l'utente (se è disponibile il servizio `identd` secondo l'RFC 1413) relativi alla connessione da remoto; `log_on_success` permette anche di registrare lo stato di uscita del server (con `EXIT`), la durata della connessione (con `DURATION`) ed il numero identificativo del processo (con `PID`).

Infine l'ultimo attributo, `per_source`, permette di stabilire un numero massimo per le connessioni da un singolo IP, una misura di protezione per limitare eventuali *denial of service*. Un elenco dei principali attributi che possono essere specificati nella sezione `defaults`, con la relativa descrizione, è riportato in tab. 8.1, l'elenco completo può essere trovato nelle pagine di manuale, accessibili con `man xinetd.conf`.

Come accennato il programma è in grado sia di implementare direttamente dei controlli di accesso, che di utilizzare quelli eventualmente specificati tramite i file di controllo dei *TCP wrappers* (vedi sez. 8.1.5). Si tenga presente che l'uso dei *TCP wrappers* ha di norma la precedenza sul controllo interno, per cui se si abilita l'accesso con queste direttive, ma esso è negato dai TCP wrappers, questi ultimi avranno la meglio. I due attributi `only_from` e `no_access` sono in grado di specificare direttamente le stesse condizioni all'interno del file di configurazione di `xinetd`. In questo caso le sintassi supportate sono sia le forme dotted decimal (interpretando gli zeri finali come indirizzi di rete), che quelle CIDR, sia quelle espresse con nomi a dominio.

Come già accennato ciascun servizio che si vuol lanciare con `xinetd` deve essere specificato con la direttiva `service` seguita dal nome dello stesso (che specifica anche la porta che verrà utilizzata, come risultante in `/etc/services`) mentre le direttive che vi si applicano vanno specificate successivamente fra parentesi graffe. Se il nome non è presente in `/etc/service` (ad esempio se si usa una porta arbitraria per un servizio proprio) si deve dichiarare il servizio come `UNLISTED` con l'attributo `type` (e si dovrà definire la porta con l'attributo `port`). Un possibile esempio di configurazione è il seguente:

Attributo	**Descrizione**
`instances`	numero massimo di processi lanciati per ogni servizio; prende un valore intero.
`log_type`	metodologia di registrazione dei log, su file o tramite *syslog*; prende i valori `FILE` (seguito dal nome del file su cui vengono aggiunti i messaggi) o `SYSLOG` seguito dalla *facility* da usare ed opzionalmente dalla priorità.
`log_on_success`	cosa registrare per le connessioni riuscite; prende uno o più fra `PID`, `HOST`, `USERID`, `EXIT`, `DURATION`.
`log_on_failure`	cosa registrare per le connessioni fallite, prende uno o più fra `HOST`, `USERID`, `ATTEMPT`.
`per_source`	numero massimo di connessioni per singolo IP sorgente; prende un valore intero.
`only_from`	lista delle macchine da cui è possibile accedere; prende indirizzi IP in forma dotted decimal, CIDR o nomi a dominio.
`no_access`	lista delle macchine da cui è impossibile accedere; prende indirizzi IP in forma dotted decimal, CIDR o nomi a dominio.
`access_times`	orario nel quale è possibile accedere ai servizi, prende un intervallo temporale specificato nella forma `HH:MM-HH:MM`.
`cps`	rate massimo di accesso; prende un valore decimale per indicare il limite sulle connessioni al secondo, ed un valore intero per specificare il numero di secondi da aspettare prima di accettare nuove connessioni una volta superato il limite.
`nice`	valore di *nice* (vedi sez. 1.3.4) da applicare ai demoni lanciati; prende un valore intero.
`max_load`	carico massimo oltre il quale la macchina smette di accettare connessioni; prende un valore decimale.

Tabella 8.1: Attributi specificabili in generale per tutti i servizi gestiti attraverso il super-demone `xinetd`.

xinetd.conf

```
service time
{
        disable = yes
        type            = INTERNAL
        id              = time-stream
        socket_type     = stream
        protocol        = tcp
        user            = root
        wait            = no
}
service time
{
        disable         = yes
        type            = INTERNAL
        id              = time-dgram
        socket_type     = dgram
        protocol        = udp
        user            = root
        wait            = yes
}
service nntp
{
        socket_type = stream
        wait        = no
        user        = news
        server      = /usr/sbin/leafnode
        server_args = -v
```

```
        only_from   = 192.168.0.0/24
        access_times = 08:00-17:00
}
```

In questo caso si sono definiti due servizi gestiti direttamente da `xinetd`, e cioè il servizio *time* su TCP e UDP, entrambi sono disabilitati, avendo `disable` impostato su `yes`. Si noti come siano stati differenziati attraverso la presenza di un argomento `id`. Inoltre con `socket_type` si è specificato il tipo di socket da usare, e con `protocol` il relativo protocollo. Il campo `type` dice che il servizio è fornito internamente, mentre `user` indica che verrà eseguito per conto dell'utente *root*. Infine il campo `wait` indica, con lo stesso significato che ha per `inetd`, se `xinetd` deve attendere la conclusione del programma lanciato, lasciando gestire a lui ulteriori connessioni, o se deve mantenere il controllo della porta lanciando altre istanze in caso di nuove connessioni.

Attributo	**Descrizione**
`socket_type`	tipo di socket; prende gli stessi valori (`stream`, `dgram`, ecc.) dell'analogo parametro in `inetd.conf`.
`user`	utente per conto del quale è lanciato il servizio; deve essere presente in `/etc/passwd`, si può specificare un eventuale gruppo con l'attributo `group`.
`server`	pathname del programma server da lanciare; deve essere sempre specificato se il servizio non è gestito internamente.
`wait`	indica se si deve attendere o meno la conclusione del server per lanciare un'altra istanza mantenendo o meno il controllo della porta; è l'analogo dello stesso parametro in `inetd.conf` e prende i valori `yes` e `no`.
`protocol`	protocollo usato (analogo di `tcp` ed `udp` per `inetd.conf`); deve essere un nome valido in `/etc/protocols`.
`port`	porta su cui ascoltare le connessioni; deve essere specificata se si è specificato un servizio non riportato in `/etc/services` (nel caso di uso di servizi di tipo `UNLISTED`).
`type`	tipo di servizio; può assumere i valori `INTERNAL` (per servizi gestiti internamente), `RPC` per indicare che si userà un servizio RPC (vedi sez. 7.6.5), `UNLISTED` per servizi non presenti in `/etc/services` o `/etc/rpc`.
`server_args`	eventuali argomenti da passare al programma server quando viene lanciato; al contrario di `inetd` non necessita di specificare il nome del comando come argomento iniziale.
`disable`	indica se attivare il servizio, il default è disattivo; può assumere i valori `yes` e `no`.
`id`	identificatore aggiuntivo qualora si tratti di servizi diversi con lo stesso nome.
`bind`	consente di specificare l'interfaccia su cui fornire il servizio; prende l'indirizzo IP ad essa associato.
`redirect`	consente di redirigere il servizio ad un'altra macchina; prende l'indirizzo IP e la porta verso quale redirigere tutto il traffico separati da uno spazio.
`umask`	imposta la *umask* ereditata dal processo mandato in esecuzione.

Tabella 8.2: Attributi specificabili per un servizio gestito attraverso il super-demone `xinetd`.

Oltre ai due servizi interni si è abilitato anche il servizio di news; in questo caso restano specificati con lo stesso significato precedente gli argomenti `wait`, `socket_type` e `user` (anche se per quest'ultimo si è usato l'utente `news`), mentre non esistendo il servizio `news` su UDP non è stato necessario impostare `protocol`. Si è invece specificato il programma da usare come server con `server`, passando i relativi argomenti con `server_args`. Inoltre nel caso si è ristretto

l'accesso al servizio alle macchine della sottorete `192.168.0.0/24` con `only_from` e negli orari di ufficio usando `access_times`.

Oltre a quelli appena illustrati, si sono riportati i principali attributi utilizzabili in tab. 8.2. Si ricordi che anche i precedenti attributi di tab. 8.1 possono essere utilizzati, e saranno applicati solo al servizio in questione. Al solito nella pagina di manuale di `xinetd.conf` è riportato un elenco completo di tutti gli argomenti presenti e la descrizione dettagliata di ciascuno di essi.

Anche con `xinetd` è possibile creare un proprio servizio usando i comandi di shell; ripeteremo allora quanto visto con `inetd` definendo un nuovo servizio `netstat`. Dato che il servizio è previsto in `/etc/services` solo per TCP possiamo attivarlo creando in `/etc/xinetd.d` un nuovo file con un contenuto del tipo di:

/etc/xinetd.d/netstat

```
service netstat
{
        disable         = no
        socket_type     = stream
        wait            = no
        user            = root
        server          = /bin/netstat
        server_args     = -ant
}
```

e, una volta riavviato `xinetd`, potremo verificarne come prima il funzionamento, con un `telnet` sulla porta 15.

Una delle caratteristiche più utili di `xinetd` è quella che consente di utilizzarlo per redirigere le connessioni ad una porta verso un'altra macchina ed eventualmente un'altra porta. Benché questo sia possibile anche utilizzando le funzionalità del *netfilter* di Linux con `iptables`,[3] quando la redirezione deve essere fatta verso un server esterno (il caso più comune è lo spostamento di un servizio presso un diverso provider) la cosa può risultare abbastanza complessa.

Si supponga allora che si sia spostato il proprio server di posta presso uno diverso provider, e sia in corso la propagazione dei DNS (vedi sez. 9.1.1) per il nuovo indirizzo IP, in tal caso chi sta ancora utilizzando il vecchio indirizzo contatterà la vecchia macchina. Per garantire una continuità nell'accesso si potranno redirigere tutte le connessioni da questa al nuovo server con `xinetd` utilizzando le funzionalità di redirezione; in questo caso basterà installare una configurazione del tipo:

/etc/xinetd.d/popredir

```
service pop3
{
        disable         = no
        socket_type     = stream
        wait            = no
        user            = root
        redirect        = IP.DEL.NUOVO.SERVER 110
}
```

Con questa configurazione, una volta riavviato il servizio, le connessioni al servizio POP3 verranno ricevute da `xinetd` che le girerà, in maniera trasparente per l'utente, al nuovo server,

[3] per una trattazione approfondita dello stesso si consulti il cap. 2 di [SGL].

reinviando al client le risposte. Così sarà possibile fornire accesso, usando il nuovo server, anche a coloro che non hanno ancora ottenuto il nuovo indirizzo.

8.1.4 Il servizio NTP

Abbiamo già accennato in sez. 2.4.3 come sia possibile usare il comando `ntpdate` per rimettere l'orologio di una macchina, andando ad interrogare un server esterno che fornisce il servizio di *ora esatta* attraverso il protocollo NTP.[4] Si può però anche tenere costantemente aggiornato l'orologio installando direttamente un server NTP.

Il servizio NTP usa la porta 123 UDP, e prevede una organizzazione gerarchica dei server in *strati* con precisione decrescente dei tempi. La gerarchia parte dai dispositivi di strato 0, che sono quelli collegati direttamente ad una fonte primaria della misura esatta del tempo (e possono essere orologi atomici, orologi GPS o radio). Ogni volta che un server NTP viene sincronizzato su un altro server NTP, la convenzione vuole che il suo numero di strato venga aumentato di 1, per questo lo strato 0 indica la connessione diretta ad una fonte di tempo.

In genere server pubblici di strato 0 (a meno di non utilizzare un propria fonte di tempo) non sono disponibili direttamente all'utente finale, e vengono invece utilizzati dei server di strato 1, che formano quelli che più comunemente si chiamano *Time Server*. Un gruppo di questi server è fornito all'indirizzo `pool.ntp.org`; l'indirizzo viene risolto dal DNS in una serie diversa di server di strato 1, per cui si possono avere diverse fonti. Un server NTP italiano, sincronizzato sull'orologio atomico dell'Istituto Galileo Ferraris di Torino, è raggiungibile all'indirizzo `tempo.ien.it`.

In genere quando si installa un server NTP su una rete si crea un server di strato 2, che fa riferimento ad uno o più server di strato 1 per ottenere la fonte di misura dei tempi. Si può poi anche installare il servizio sulle singole macchine, nel qual caso è opportuno che queste facciano riferimento al precedente server di strato 2, e si avrebbero a quel punto server di strato 3.

In tal caso infatti, a parte il fatto di non generare inutilmente traffico di rete verso l'esterno per rivolgersi in più macchine ad un server di strato 1, è prevalente l'interesse di avere una migliore coerenza fra i tempi degli orologi delle macchine (cosa ottenibile rivolgendosi ad un server sulla propria rete), piuttosto che una maggiore "*esattezza media*" delle ore dei vari orologi. L'operazione si potrebbe ripetere creando eventuali server di strato 4 e così via fino ad un massimo di 16 strati, ma con scarsa utilità pratica.

L'utilizzo di un server NTP rispetto all'invocazione periodica di `ntpdate` ha il vantaggio che il demone è in grado di raccogliere informazioni riguardo alla deriva dell'orologio di sistema, e fornire allo stesso le necessarie informazioni per mantenere corretta l'ora in maniera costante (facendolo accelerare o rallentare per compensare le differenze rilevate), senza che diventi necessaria un sincronizzazione periodica.

Il server `ntpd` fornisce una implementazione completa della versione 4 del protocollo, pur mantenendo la compatibilità con la versione 3. Benché si possano effettuare alcune impostazioni di base direttamente con le opzioni a riga di comando, in genere il comando viene lanciato dagli

[4] il *Network Time Protocol* è uno dei più complessi disponibili su Internet e consente una sincronizzazione dei tempi molto precisa (dell'ordine dei ms in locale e delle decine di ms in remoto) nonostante tutte le incertezze che si possono avere nella trasmissione dei pacchetti; l'RFC 1305 che descrive la versione 3 del protocollo supera il centinaio di pagine, la versione quattro, che è pienamente supportata da `ntpd`, è descritta dall'RFC 5905 ed è stata standardizzata (RFC 2030) una versione semplificata del protocollo, SNTP, utilizzata quando non è necessaria la precisione del protocollo completo.

script di avvio, e legge la sua configurazione da /etc/ntp.conf (o dal file specificato con l'opzione -c). Qualora si avessero problemi si può lanciare direttamente il server in modalità di debug con l'opzione -d,[5] in tal caso il programma non si distaccherà dal terminale e stamperà tutti i messaggi.

Direttiva	**Opzione**	**Significato**
driftfile	-f	prende come parametro il file dove verranno scritte le informazioni relative alla deriva dell'orologio di sistema, usate per le relative correzioni.
statsdir	-s	indica la directory su cui vengono salvate le informazioni statistiche.
server	–	indica un server di riferimento da cui ottenere la sincronizzazione, passato come argomento per nome a dominio o indirizzo IP, la direttiva può essere ripetuta più volte per indicare più server.
restrict	–	imposta le regole di accesso al server, prende come primo parametro un indirizzo IP di una macchina o di una rete (nel qual caso deve essere seguita dalla sottodirettiva mask e dalla relativa *netmask*) cui seguono le restrizioni da applicare. Deve essere specificata almeno una volta per indicare una regola di default (in tal caso si può usare la parola chiave default al posto dell'indirizzo).

Tabella 8.3: Le principali direttive di configurazione di ntp.conf ed eventuali corrispondenti opzioni di ntpd.

Il formato del file prevede la solita convenzione di ignorare righe vuote o inizianti per "#", ogni riga è costituita da una direttiva seguita da uno o più argomenti, separati da spazi, e non si può proseguire sulla riga successiva. La direttiva più importante, ed in genere l'unica che serve modificare, è server, usata per indicare a quali server di strato superiore ci si deve rivolgere per ottenere la sincronizzazione.

Una seconda direttiva, da utilizzare quando si intende impostare un server NTP locale ad uso di altre macchine, è restrict, che consente di indicare a quali IP il server può rispondere per le richieste di sincronizzazione. Le altre principali direttive di configurazione di /etc/ntp.conf sono riportate in tab. 8.3, insieme alle opzioni che permettono di impostare il valore di alcune di esse se si lancia il demone a riga di comando.

Un estratto del file di configurazione installato su una Debian Sarge per un server di strato 2 è il seguente, dove si è indicato un server di strato 1 da usare come riferimento e si è abilitato l'accesso dalla propria rete locale:

/etc/ntp.conf

```
driftfile /var/lib/ntp/ntp.drift
statsdir /var/log/ntpstats/
statistics loopstats peerstats clockstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable
server pool.ntp.org
restrict default kod notrap nomodify nopeer noquery
restrict 127.0.0.1 nomodify
restrict 192.168.1.0  mask  255.255.255.0 nomodify notrap
```

[5]sempre che il relativo supporto sia stato abilitato nella compilazione del programma, nel qual caso si può ripetere l'opzione più volte per avere messaggi di debug più dettagliati.

La documentazione delle varie opzioni e delle principali direttive di configurazione del server è disponibile nella pagina di manuale, accessibile con `man ntpd`, ma per la documentazione completa occorre fare riferimento al sito del progetto, su `http://www.ntp.org/documentation.html`.

Una volta installato un server NTP se ne può controllare il funzionamento con il comando `ntpq`. Se eseguito senza argomenti il comando fornisce una interfaccia testuale ed opera in modalità interattiva, nella quale si possono scrivere direttamente i vari comandi interni che interrogano il server NTP, che possono essere invece forniti sulla riga di comando come parametri per l'opzione `-c`. Di default il comando si collega al server NTP su `localhost`, ma si può indicare un server diverso passandogli l'indirizzo come argomento.

In genere l'uso più comune del comando è invocarlo con l'opzione `-p` che richiede la stampa dei server a cui si sta facendo riferimento, in modo da verificare che la sincronizzazione stia funzionando. Si può usare poi l'opzione `-n` per evitare la risoluzione inversa degli indirizzi IP. In tal caso il comando stampa un elenco dei server che sono stati contattati con qualcosa del tipo:

```
root@monk:~# ntpq -np
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
+5.249.146.81    193.11.166.36    2 u  264 1024  377   43.117    0.586   2.716
-85.18.189.242   62.48.53.90      3 u  563 1024  337   47.407   63.370  64.227
*212.45.144.3    193.204.114.232  2 u   11 1024  377   31.621   -0.626   0.604
+146.48.81.102   193.204.114.233  2 u  963 1024  375   47.269    0.366   0.883
```

che ci mostra come siano stati contattati 4 server fino ad un massimo di 963 secondi fa (la colonna `when` indica l'ultima interrogazione), di questi viene marcato con un "`*`" iniziale il server usato al momento come riferimento primario, mentre con un "`+`" si indicano i candidati ad essere ulteriori riferimenti e con "`-`" quelli scartati.

8.1.5 I TCP wrappers

Per molto tempo gran parte dei servizi di rete, nati in un periodo in cui Internet era una rete costituita principalmente da istituzioni ed enti di ricerca, ed in cui il numero delle persone che potevano utilizzarla era molto limitato e facilmente controllabile, non sono stati forniti di nessun tipo di controllo degli accessi, permettendo a chiunque di collegarsi ad essi, qualunque fosse la sua macchina o il suo indirizzo IP.

Con l'espandersi della rete e la possibilità di accessi non voluti o *maliziosi*, è diventato sempre più importante poter effettuare un controllo degli accessi. Per questo Wietse Wenema, un esperto di sicurezza, ha creato un insieme di librerie e programmi, chiamati *TCP wrappers*, che consentono di realizzare un controllo degli accessi permettendo il collegamento ai servizi da essi protetti solo da parte di certi indirizzi o reti.[6]

Le modalità con cui si possono utilizzare le capacità di filtraggio dei *TCP wrappers* sono sostanzialmente due. La prima è quella più semplice che vede l'uso del programma `tcpd`, che fa da "*involucro*" (in inglese *wrapper*, da cui il nome) all'esecuzione di altri programmi. Esso viene di norma utilizzato attraverso il demone `inetd` come illustrato in sez. 8.1.2 per lanciare altri programmi. Il comando infatti effettua un controllo sul socket associato alla connessione e se

[6] in sostanza questi programmi leggono dal socket associato ad una connessione l'indirizzo sorgente da cui essa è stata effettuata, ed applicano delle restrizioni sui possibili valori di quest'ultimo.

l'accesso è consentito lancia il demone specificato come argomento; in questo modo è possibile realizzare un controllo di accesso anche per programmi che non hanno alcun supporto per questa funzionalità.

La seconda modalità è quella che prevede l'uso, direttamente all'interno del demone che fornisce il servizio, delle librerie di controllo dei *TCP wrappers*, in questo modo è il programma stesso che diventa in grado di usarne le funzionalità senza dover essere lanciato da un programma esterno. Alcuni demoni che fanno uso di questa modalità sono quelli relativi a servizi come SSH, LDAP o NFS. In entrambi i casi le connessioni ai servizi controllati con i *TCP wrappers* vengono registrate sul sistema del *syslog*, in modo da lasciare traccia di eventuali tentativi di accesso non autorizzati, dopo di che vengono eseguiti i vari controlli che sono stati richiesti.

Il controllo di accesso dei *TCP wrappers* è gestito attraverso due file, `/etc/hosts.allow` ed `/etc/hosts.deny`, che contengono le regole di accesso. Il primo file, come suggerisce il nome, elenca le regole che consentono l'accesso, il secondo quelle che lo negano.

Si tenga presente che il funzionamento dei *TCP wrappers* è tale che prima viene controllato `hosts.allow`, e se una regola corrisponde l'accesso è garantito e la procedura di controllo finisce immediatamente. Altrimenti viene controllato `hosts.deny` e se una regola corrisponde l'accesso è negato. Se non si trova nessuna corrispondenza (o entrambi i file sono vuoti o non esistono), l'accesso è consentito.

Dato che è buona norma in una politica di sicurezza bloccare tutto per default per poi consentire soltanto quanto necessario, in genere quello che si fa è negare tutti gli accessi in `hosts.deny` e garantire poi solo quelli effettivamente voluti in `hosts.allow`. Per questo un esempio tipico del contenuto di `hosts.deny` è il seguente:

/etc/hosts.deny

```
# /etc/hosts.deny: list of hosts that are _not_ allowed to access the system.
#                  See the manual pages hosts_access(5), hosts_options(5)
#                  and /usr/doc/netbase/portmapper.txt.gz
#
ALL: ALL
```

in cui si blocca l'accesso a tutti i servizi da qualunque provenienza.

La sintassi delle regole da utilizzare in questi file già si intravede da questo primo esempio; al solito righe vuote e tutto quello che segue un `#` viene ignorato, mentre ogni riga non vuota deve avere la forma generica:

```
lista dei server: lista dei client : comando shell
```

con tre campi di configurazione separati dal carattere "`:`", di questi il terzo è opzionale e può essere omesso (ed in effetti è scarsamente utilizzato). I primi due campi, come riportato, consentono l'uso di una lista di valori, da specificare separati da spazi.

Per capire meglio la sintassi prendiamo un secondo esempio, più significativo del precedente, per un possibile contenuto di `hosts.allow` che, una volta bloccato tutto con il precedente `hosts.deny`, consenta comunque di accedere ad alcuni servizi:

/etc/hosts.allow

```
# /etc/hosts.allow: list of hosts that are allowed to access the system.
#                   See the manual pages hosts_access(5), hosts_options(5)
#                   and /usr/doc/netbase/portmapper.txt.gz
```

```
#
sshd:     ALL
leafnode: 127.0.0.1
portmap:  127.0.0.1 192.168.1.
mountd:   127.0.0.1 192.168.1.
statd:    127.0.0.1 192.168.1.
lockd:    127.0.0.1 192.168.1.
rquotad:  127.0.0.1 192.168.1.
```

L'esempio riporta alcuni criteri di accesso per diversi servizi: il primo concerne l'uso di SSH (che tratteremo in sez. 8.3) per l'accesso da remoto alla riga di comando, che viene gestito dal programma `sshd`; il secondo è il servizio che permette di tenere un server di news in locale, gli altri sono i vari server necessari al funzionamento di NFS (si veda sez. 8.4.1). L'esempio consente l'accesso generico ad SSH da qualunque indirizzo IP, l'accesso a NFS solo dalla rete locale, e l'uso del server di news solo dalla macchina stessa.

In generale per il primo campo serve ad indicare il nome (o i nomi, se sono più di uno) del programma che gestisce il servizio sui cui si vuole applicare il controllo di accesso. Occorre fare attenzione, perché nelle versioni meno recenti dei *TCP wrapper* il nome da indicare nella prima colonna è proprio quello del programma che fornisce il servizio, e non quello della porta che si usa invece in `inetd.conf`; le versioni più recenti consentono anche di utilizzare il nome del servizio (o il relativo numero di porta).

Nella secondo campo si indicano invece gli indirizzi IP o le reti a cui si vuole consentire l'accesso. Per le reti in genere si può usare sia la notazione numerica che quella dei nomi a dominio, e si può usare il carattere "*" per raggruppare indirizzi; si può anche specificare una maschera di rete con un indirizzo del tipo:

```
131.155.72.0/255.255.254.0
```

e specificare la lista degli indirizzi usando un file, dando il pathname assoluto dello stesso.

Il formato e la lista completa delle funzionalità che sono controllabili tramite questi due file è riportato nella pagina di manuale ad essi associata, accessibile con `man 5 hosts_access`. Si tenga comunque conto che alcuni servizi supportano solo un sottoinsieme delle funzionalità definite in generale, come accade ad esempio per il server NFS del kernel, ci sono infatti casi in cui questi file vengono utilizzati direttamente, in maniera indipendente dalle librerie dei *TCP wrapper*, con un sottoinsieme delle capacità di queste ultime.

Per una migliore gestione dei *TCP wrapper* col relativo pacchetto vengono forniti anche dei programmi di utilità che permettono di verificare la configurazione ed effettuare dei controlli di accesso. Il primo programma è `tcpdchk` che esegue un controllo delle regole di accesso impostate con `hosts.allow` e `hosts.deny`, confrontandole anche con i servizi attivati in `inetd.conf`. Il comando riporta tutti gli eventuali problemi rilevati, a partire da un uso scorretto di wildcard e indirizzi, servizi che non sono riconosciuti, argomenti o opzioni non validi, ecc. Così ad esempio potremo avere:

```
root@gont:~# tcpdchk
warning: /etc/hosts.allow, line 15: apt-proxy: no such process name in /etc/inetd.conf
```

in corrispondenza ad una regola di accesso rimasta aperta per un servizio che in seguito è stato rimosso.

Il comando permette di controllare i file `hosts.allow` e `hosts.deny` nella directory corrente invece che sotto `/etc` usando l'opzione `-d`, mentre si può specificare un diverso file per `inetd.conf` con l'opzione `-i`, la documentazione completa è al solito disponibile con `man tcpdchk`.

Il secondo comando di controllo è `tcpdmatch` che permette di verificare il comportamento dei TCP wrapper per una specifica richiesta da un servizio. Il comando richiede due parametri, il primo che specifichi il servizio che si vuole controllare ed il secondo la stazione da cui si intende effettuare l'accesso. Il comando eseguirà una scansione delle regole e riporterà i risultati. Ad esempio potremo richiedere:

```
root@gont:~# tcpdmatch sshd oppish
warning: sshd: no such process name in /etc/inetd.conf
warning: oppish: hostname alias
warning: (official name: oppish.earthsea.ea)
client:   hostname oppish.earthsea.ea
client:   address  192.168.1.168
server:   process  sshd
matched:  /etc/hosts.allow line 14
access:   granted
```

che controlla l'accesso al servizio SSH (si noti che si deve specificare il nome del programma che esegue il servizio) da parte della macchina `oppish`, trovando che questo è consentito dalla riga 14 del file `hosts.allow`. Il comando rileva anche che il servizio non è lanciato attraverso `inetd` e quale è l'IP effettivo della macchina.

8.2 L'assegnazione dinamica degli indirizzi IP

In questa sezione tratteremo vari protocolli (ed i relativi programmi di utilizzo) che vengono usati per l'assegnazione automatica degli indirizzi IP all'interno di una rete locale. Il primo, il RARP è stato il primo protocollo usato a questo scopo, ed è sostanzialmente in disuso essendo sostituito dagli altri due. Il secondo, il BOOTP, nasce per distribuire via rete immagini di avvio, mentre l'ultimo, il DHCP, ricomprende tutte queste esigenze all'interno di un unico servizio, e lo tratteremo con maggiori dettagli.

8.2.1 I protocolli RARP, BOOTP e DHCP

Il protocollo RARP (sigla che sta per *Reverse Address Resolution Protocol*), è un protocollo elementare, definito nell'RFC 903 che, come il nome stesso indica, esegue il compito inverso rispetto al protocollo ARP già visto in sez. 7.6.4. Il protocollo è implementato direttamente a livello di collegamento fisico (cioè nella maggior parte dei casi su ethernet) e serve ad ottenere, dato un *MAC address* sulla rete, l'indirizzo IP ad esso associato.

Lo scopo del protocollo era principalmente quello di fornire un meccanismo automatico per l'assegnazione di un indirizzo IP ad una macchina in fase di avvio; questa esegue (posto che il kernel disponga del relativo supporto, che su Linux deve essere opportunamente abilitato) una richiesta RARP su ethernet ed utilizza l'indirizzo IP fornito come risposta per configurare l'interfaccia di rete da cui ha eseguito la richiesta; la cosa presuppone ovviamente la presenza nel suddetto tratto di rete di un server che fornisca la risposta.

Tutto ciò oggi, in presenza di altri protocolli più sofisticati in grado di eseguire sia questo che altri compiti, ha solo interesse storico (considerato anche che sono assai pochi ormai i sistemi operativi che lo supportano). Per chi si trovasse in situazioni in cui per compatibilità con vecchi sistemi viene ancora usato questo metodo per l'assegnazione degli indirizzi, ci limitiamo a citare la possibilità di compilare il kernel per utilizzare questo supporto.

Il protocollo BOOTP (il nome sta per *Bootstrap Protocol*), è definito nell'RFC 951 e nasce come protocollo per gestire l'avvio automatico delle macchine. Rispetto al precedente RARP è basato su IP e UDP (e non direttamente sul collegamento fisico) e non si limita alla ricerca di un indirizzo IP da assegnare automaticamente ad una scheda di rete, ma prevede anche l'invio delle informazioni necessarie ad ottenere un file da usare come immagine del sistema operativo per l'avvio e l'indirizzo di una macchina a cui richiedere quest'ultimo.

Il protocollo BOOTP nasce principalmente per fornire un meccanismo con cui macchine senza disco possono eseguire un avvio del sistema operativo via rete, ottenendo quest'ultimo, (cioè i vari file contenenti i dati necessari, nel caso di Linux l'immagine del kernel ed un eventuale *ramdisk*) da un opportuno server. In genere il trasferimento dei dati avviene tramite il protocollo TFTP,[7] ma in teoria possono essere utilizzati anche altri protocolli. In questo caso è compito del BIOS della macchina implementare la parte client del protocollo (sia di BOOTP che di TFTP, se poi viene usato quest'ultimo) in modo da ottenere poi l'immagine del sistema che verrà caricata in memoria ed eseguita.

Come sottoinsieme delle funzionalità del protocollo BOOTP c'è ovviamente anche quella della assegnazione automatica di un indirizzo IP dato che il trasferimento dell'immagine del sistema avviene sempre con un altro protocollo che necessita di IP. Per questo motivo il protocollo può essere usato per distribuire gli indirizzi su una rete. Di nuovo qualora questa funzionalità fosse necessaria per l'avvio, ad esempio per l'avvio di un sistema con la directory radice su NFS, occorrerà compilare il kernel con il relativo supporto, analogamente a quanto si farebbe per l'uso di RARP.

Il supporto per la parte server del protocollo è disponibile con il programma `bootpd` (su Debian il pacchetto è `bootp`) che implementa tutte le funzionalità definite nell'RFC 951 e pure le estensioni dei successivi RFC 1532 e RFC 1533. In genere il servizio viene avviato tramite `inetd`, ma può essere eseguito anche in modalità *standalone*.

Il comando prevede come argomento il file da cui leggere le impostazioni, che può essere omesso, nel qual caso verrà utilizzato il default che è `/etc/bootptab`; quest'ultimo contiene tutte le definizioni dei vari parametri da usare per ciascun client nella forma:

```
hostname:tg=value... :tg=value... :tg=value. ...
```

dove `hostname` indica il nome associato ad un client cui seguono le assegnazioni dei parametri che sono identificati da una etichetta di due caratteri, separata con l'uso del carattere "`:`".

Un elenco delle etichette dei principali parametri usati da `bootptab` è riportato in tab. 8.4, al solito l'elenco completo e tutti i dettagli sono nella relativa pagina di manuale, accessibile con `man bootptab`.

Il protocollo più utilizzato per la configurazione automatica della rete è il DHCP, sigla che sta per *Dynamic Host Configuration Protocol*, che al giorno d'oggi ha soppiantato in maniera

[7] il *Trivial File Transfer Protocol* è un protocollo elementare per il trasferimento di file basato su UDP, che è utilizzabile in programmi estremamente semplificati come quelli che sono contenuti nel BIOS di una macchina.

Tag	Descrizione
`bf`	file di avvio.
`dn`	nome del dominio.
`ds`	lista dei server DNS.
`gw`	lista dei gateway.
`ha`	indirizzo fisico (*MAC address*) della macchina.
`hd`	home directory per il file di avvio.
`hn`	*hostname* da inviare al client.
`ip`	indirizzo IP da assegnare.
`rp`	pathname della directory da montare come radice.
`sa`	indirizzo del server TFTP da usare.
`sm`	maschera di rete della macchina.
`td`	directory radice per il server TFTP.

Tabella 8.4: Etichette dei parametri assegnabili in `bootptab`.

praticamente completa sia RARP che BOOTP. Oltre ai servizi previsti dal precedente BOOTP, del quale implementa, in maniera compatibile all'indietro, tutte le funzionalità, il protocollo supporta anche l'assegnazione dinamica degli indirizzi. La sua descrizione completa è disponibile nell'RFC 2131, che ne definisce tutte le caratteristiche.

A differenza di BOOTP lo scopo principale di DHCP, come espresso dal nome stesso, è quello della configurazione dinamica delle macchine in una rete locale. Uno dei vantaggi di questo protocollo è che è in grado di fornire parecchie informazioni in più rispetto a quelle elementari fornite da BOOTP, rendendo possibile la configurazione automatica di tutte le funzionalità della rete per le macchine di una LAN.

La peculiarità di DHCP è comunque il supporto per la gestione dinamica degli indirizzi. Con BOOTP infatti l'assegnazione di un indirizzo IP è fatta staticamente sul server dove per ogni client deve essere configurato un indirizzo. Il client si limita a fare una richiesta e configurare la rete in base alla risposta (o fallisce se il server non risponde). Non viene gestito il caso di una assegnazione temporanea all'interno di un pool di indirizzi, cioè il caso tipico di quando al computer di un ospite in visita deve essere assegnato un indirizzo temporaneo all'interno della propria rete.

Una assegnazione di questo tipo comporta ovviamente una serie di problemi in più rispetto all'assegnazione statica, ed in particolare l'assegnazione degli indirizzi si scontra con le problematiche relative alla presenza discontinua od occasionale di macchine all'interno della rete, alla necessità di dover gestire la riassegnazione di indirizzi usati in precedenza, di accorgersi quando un indirizzo non è più in uso, e di evitare di assegnare lo stesso indirizzo a macchine diverse.

Per risolvere questo tipo di problematiche DHCP prevede una comunicazione non occasionale fra il client, che resta attivo anche dopo la configurazione dell'indirizzo, ed il server e tre diverse modalità di assegnazione degli indirizzi, dette rispettivamente *statica*, *automatica* e *dinamica*.

La configurazione *statica* è sostanzialmente la stessa di BOOTP, in cui si associa staticamente un indirizzo ad una specifica macchina sulla base del *MAC address* della stessa. Con la configurazione *automatica* invece viene assegnato in maniera permanente un indirizzo IP ad ogni macchina che ne fa richiesta, ma non è necessario che l'amministratore imposti la corrispondenza come nel caso precedente. Questo può andare bene nel caso di macchine sempre presenti sulla rete, ma non è ovviamente adatto al caso di presenze occasionali.

Per questo la modalità che alla fine viene utilizzata di più è quella della configurazione *dina-*

mica, in cui ad ogni assegnazione è associato anche un tempo di permanenza, il cosiddetto *lease*, passato il quale, se il client non ha chiesto un prolungamento, l'assegnazione viene cancellata. In questo modo diventa possibile gestire la presenza occasionale di macchine sulla rete, assegnando un indirizzo libero che tornerà automaticamente disponibile dopo un certo periodo di inutilizzo, o che potrà essere rilasciato esplicitamente quando segnalato dal client stesso.

Il protocollo DHCP prevede che sia sempre il client a contattare il server; all'avvio il client invia dei pacchetti in *broadcast* su UDP,[8] e alla ricezione di una richiesta il server invia un pacchetto di risposta contenente le varie informazioni di configurazione,[9] che prevedono l'indirizzo IP da assumere, quello del *default gateway*, l'indirizzo di eventuali server DNS ed altre informazioni relative alla rete, ma soprattutto, qualora si usi la configurazione dinamica, il periodo di *lease* per il quale l'assegnazione dell'indirizzo sarà considerata valida.

8.2.2 Uso del servizio DHCP dal lato client

Per poter usufruire dei servizi di un server DHCP esistono vari programmi; i più comuni e diffusi sono `pump` e `dhclient`, di quest'ultimo esistono poi due versioni, a seconda che si usi la versione 2 o la versione 3 del protocollo, anche se ormai in uso resta praticamente solo quest'ultimo.

Il programma `pump` è un client sia per DHCP che per BOOTP. Il suo uso è immediato, basta invocarlo specificando l'interfaccia che si vuole configurare in maniera automatica con l'opzione `-i`, una invocazione tipica è qualcosa del tipo:

```
pump -i eth0
```

anche il programma può essere lanciato automaticamente all'avvio del sistema, quando si configura la rete per eseguire la configurazione tramite DHCP.

Il comando prende una serie di altre opzioni che consentono di scartare le informazioni ottenute dal server (come i DNS o il *default gateway*) o controllare lo stato del servizio, al solito per i dettagli si può fare riferimento alla pagina di manuale. Il programma può anche essere controllato tramite un file di configurazione, `/etc/pump.conf`, il cui formato è descritto nella relativa pagina di manuale, accessibile con `man pump`.

Una volta ricevuta risposta da un server, il comando utilizza le informazioni ricevute per configurare la rete. Queste in genere comprendono, oltre l'indirizzo IP da assegnare, anche il dominio locale, gli indirizzi dei server DNS e quello del *default gateway*, pertanto il comando non si limiterà ad assegnare un indirizzo all'interfaccia specificata, ma creerà anche un opportuno `resolv.conf` ed inserirà la *default route* nella tabella di routing. Qualora il server non dia nessuna risposta invece il comando fallirà, così come la configurazione automatica della rete.

Il secondo programma che viene utilizzato come client per la configurazione automatica della rete è `dhclient`. Questo è parte della implementazione di riferimento del protocollo DHCP fatta

[8] ci si può chiedere come possa essere utilizzato UDP, che presuppone IP, prima che la macchina abbia un indirizzo IP con il quale contattarla; quello che accade è che i pacchetti vengono inviati con IP sorgente nullo (indirizzo `0.0.0.0` e porta 68), ed in *broadcast* come IP di destinazione (indirizzo `255.255.255.255` e porta 67) con al loro interno il *MAC address* del richiedente, che così può essere raggiunto dal server.

[9] in realtà il meccanismo è più complesso, il protocollo prevede che il client invii un pacchetto di tipo *dhcp discovery*, che in sostanza chiede se esiste un server DHCP disponibile; a questo i server presenti risponderanno con dei pacchetti di offerta (di tipo *dhcp offer*) con una proposta di indirizzo IP e di *lease*, ed il client potrà scegliere quello che preferisce (in genere in base alla maggior durata del *lease*) rispondendo con un pacchetto di tipo *dhcp request*; la negoziazione si chiude alla risposta del server con un pacchetto *dhcp ack*, senza il quale il procedimento ricomincia da capo.

dall'Internet Software Consortium, ed ad oggi viene utilizzato nella versione 3, anche se su alcune distribuzioni molto vecchie si può ancora trovare la versione 2 del programma.

L'invocazione manuale del programma è analoga a quella di `pump`, ma è necessario specificare nessuna opzione per indicare una interfaccia. Se non si passano argomenti il programma esamina le interfacce presenti, scarta quelle che non supportano il *broadcast*, e poi cerca di configurarle tutte; altrimenti si può specificare la lista delle interfacce da configurare con qualcosa del tipo:

```
dhclient eth1 eth2
```

il comando supporta tre opzioni, `-p` consente di specificare l'uso di una porta diversa da quella standard, `-d` che esegue il programma in modalità di debug mantenendolo agganciato al terminale (si tenga presente che anche il client DHCP lavora come demone, dovendo gestire la scadenza dei *lease*) e `-e` che causa l'uscita del programma se questo non riesce a configurare l'interfaccia entro un certo tempo.

Il comportamento di `dhclient` è controllato dal file `dhclient.conf` (che a seconda delle versioni può essere installato sotto `/etc/dhcp3/` o sotto `/etc/dhcp/`) riguardo ad aspetti come le temporizzazioni, le informazioni richieste al server, la presenza di valori che devono sovrascrivere o essere aggiunti a quelli presenti in eventuali risposte o utilizzati come default qualora questo non risponda.

Il formato del file prevede al solito che le righe vuote e tutto quanto segue il carattere "`#`" venga ignorato. Le direttive sono di due tipi, quelle elementari, che specificano delle caratteristiche singole, hanno la forma di una parola chiave seguita da uno o più valori e sono terminate dal carattere "`;`". Quelle complesse possono specificare più caratteristiche e prevedono una parola chiave seguita da un eventuale parametro e da un blocco di ulteriori direttive semplici (nella forma precedente) che viene delimitato da parentesi graffe. In generale spaziature e ritorni a capo vengono ignorati, ed i nomi delle direttive sono *case insensitive*.

Un esempio di `dhclient.conf`, preso dalla versione installata su una Debian Squeeze (nella versione 3 del client) è il seguente estratto, dove come si può notare la maggior parte delle direttive sono commentate:

dhclient.conf

```
# Configuration file for /sbin/dhclient, which is included in Debian's
#       dhcp3-client package.
#
# This is a sample configuration file for dhclient. See dhclient.conf's
#       man page for more information about the syntax of this file
#       and a more comprehensive list of the parameters understood by
#       dhclient.
#
# Normally, if the DHCP server provides reasonable information and does
#       not leave anything out (like the domain name, for example), then
#       few changes must be made to this file, if any.
#

option rfc3442-classless-static-routes code 121 = array of unsigned integer 8;

#send host-name "andare.fugue.com";
#send dhcp-client-identifier 1:0:a0:24:ab:fb:9c;
#send dhcp-lease-time 3600;
#supersede domain-name "fugue.com home.vix.com";
```

```
#prepend domain-name-servers 127.0.0.1;
request subnet-mask, broadcast-address, time-offset, routers,
        domain-name, domain-name-servers, domain-search, host-name,
        netbios-name-servers, netbios-scope, interface-mtu,
        rfc3442-classless-static-routes, ntp-servers;
#require subnet-mask, domain-name-servers;
#timeout 60;
#retry 60;
#reboot 10;
#select-timeout 5;
#initial-interval 2;
#script "/etc/dhcp3/dhclient-script";
#media "-link0 -link1 -link2", "link0 link1";
#reject 192.33.137.209;
...
```

In genere non è necessario modificare questo file, che può anche essere lasciato vuoto. Nell'esempio precedente la principale direttiva utilizzata è `request` che serve ad indicare quale informazioni si richiedono al server DHCP. Un'altra direttiva che può essere utilizzata è `send` che consente di inviare informazioni al server, come ad esempio l'*hostname* (vedi sez. 7.4.3) che può essere utilizzato da quest'ultimo per aggiornare la lista delle macchine presenti sulla rete. Un elenco delle principali direttive è riportato in tab. 8.5, l'elenco completo è al solito disponibile sulla pagina di manuale.

Direttiva	**Significato**
`request`	elenca la lista delle opzioni DHCP (vedi tab. 8.8) che il client richiede al server.
`timeout`	indica il tempo massimo oltre il quale il client stabilisce che non ci sono server raggiungibili.
`send`	indica una opzione DHCP (vedi tab. 8.8) ed il relativo valore che il client invia al server.
`script`	specifica lo script usato dal client per configurare l'interfaccia, di default viene usato `/etc/dhcp3/dhclient-script`.
`reject`	specifica l'indirizzo IP di un server da cui non devono essere accettate risposte.
`interface`	permette di impostare direttive di configurazione specifiche per una interfaccia; richiede come argomento il nome della interfaccia in questione ed a seguire il blocco delle direttive da applicare.

Tabella 8.5: Principali direttive del file `dhclient.conf`.

Si tenga presente che le operazioni di `dhclient` non sono limitate alla configurazione iniziale della rete; il protocollo infatti prevede che il client debba ricontattare periodicamente il server per prolungare un *lease* che sta scadendo, mantenendo l'associazione con il relativo indirizzo IP, o per riacquisire le informazioni di configurazione, o per notificare la cessazione dell'uso di un indirizzo IP in caso di uscita.

Per la gestione della assegnazione dinamica degli indirizzi `dhclient` mantiene una lista dei *lease* ottenuti nel file `dhclient.leases` (in genere questo file si trova sotto `/var/lib/dhcp/`) in modo da tenerne traccia anche se si sono effettuati riavvii del sistema o se il server ha avuto un crash. Il formato del file è descritto nella relativa pagina di manuale, accessibile con `man dhclient.lease`. Anche questo file viene letto all'avvio ed i suoi dati vengono utilizzati qualora un server DHCP sia irraggiungibile.

In questo caso quando il server torna disponibile i vecchi *lease* non ancora scaduti vengono ricontrollati e, se trovati validi, riutilizzati. Il client si cura inoltre di aggiornare `dhclient.leases` tutte le volte che ottiene una nuova associazione con un nuovo *lease*, e per evitare che il file cresca indefinitamente crea periodicamente una nuova versione del file contenente solo i *lease* attivi al momento, salvando la vecchia come `dhclient.leases~`.

8.2.3 La configurazione di un server DHCP

Sul lato server il protocollo DHCP viene realizzato tramite un apposito demone, `dhcpd`, che si incarica di ricevere le richieste e fornire le risposte. Il server viene lanciato dal relativo script di avvio (che dipende dalla distribuzione), ma può anche essere avviato a mano. In quel caso si può usare l'opzione `-d` per eseguire il server interattivamente in modalità di debug. Un'altra opzione è `-p` che permette di usare una porta diversa da quella di default (la 67 su UDP); per le altre opzioni si può fare riferimento alla pagina di manuale.

Come il client anche il server mantiene una lista delle assegnazioni effettuate e dei relativi *lease* in un apposito file, `dhcpd.leases` (sempre sotto `/var/lib/dhcp/`), così da poter tenere traccia dello stato del sistema anche in caso di riavvio, anche il formato di questo file è descritto nella pagina di manuale accessibile con `man dhcpd.leases`. Il server scrive in questo file ogni nuovo *lease* assegnato e ne cura la rotazione periodica come fa `dhclient` per `dhclient.leases`.

Il funzionamento del server è controllato da un file di configurazione che è `dhcpd.conf` (che di nuovo può essere installato sotto `/etc/dhcp/` o `/etc/dhcp3/`). Un esempio del contenuto di questo file è il seguente:

dhcpd.conf

```
#
# Sample configuration file for ISC dhcpd for Debian
#

# option definitions common to all supported networks...
option domain-name "earthsea.ea";
option domain-name-servers gont.earthsea.ea;

option subnet-mask 255.255.255.0;
default-lease-time 600;
max-lease-time 7200;

subnet 192.168.1.0 netmask 255.255.255.0 {
  range 192.168.1.32 192.168.1.63;
  option broadcast-address 192.168.1.255;
  option routers gont.earthsea.ea;
}

host oppish {
  hardware ethernet 08:00:07:26:c0:a5;
  fixed-address oppish.earthsea.ea;
}
```

Il formato del file è identico a quello di `dhclient.conf` e come per questo il contenuto è divisibile sommariamente in due categorie di direttive, quelle che prevedono la specificazione di semplici parametri e quelle più complesse, dette dichiarazioni, che contengono blocchi di altre

direttive. In generale il file prevede una serie di direttive iniziali, che servono ad impostare i valori che vengono utilizzati come default. Questi possono essere modificati a valori diversi all'interno di dichiarazioni più specifiche.

Un esempio di direttive semplici sono le righe iniziali del precedente esempio che nel caso sono state utilizzate per impostare alcune opzioni DHCP a livello generale, come il dominio di riferimento, il server DNS, o la maschera di rete. In questo caso si usa la direttiva `option`, seguito dal nome dell'opzione DHCP che si intende impostare (le principali sono riportate in tab. 8.8) ed dal relativo valore.

Altre due direttive semplici sono `default-lease-time`, che imposta il valore di default (nel caso 600 secondi) del *lease* delle risposte, e `max-lease-time` che imposta valore massimo che può essere assegnato ad un *lease*. Un elenco delle principali direttive per la dichiarazione di parametri è illustrato in tab. 8.6.

Direttiva	**Significato**
`option`	imposta il valore di una opzione DHCP, prende come argomento il nome dell'opzione (vedi tab. 8.8), seguito dagli eventuali valori.
`default-lease-time`	imposta il valore di default del tempo di *lease* inviato a un client se questo non ha fatto una richiesta specifica.
`max-lease-time`	imposta il valore massimo del tempo di *lease* che può essere inviato ad un client, indipendentemente da quanto esso possa avere richiesto.
`range`	imposta un intervallo di indirizzi da assegnare dinamicamente.
`fixed-address`	imposta un indirizzo (o più) da assegnare staticamente, viene in genere utilizzato all'interno di una dichiarazione `host` insieme a `hardware address`.
`hardware address`	indica l'indirizzo hardware (in genere un *MAC address*) cui assegnare un indirizzo statico.
`include`	legge il contenuto di un altro file come se questo fosse stato incluso nella configurazione.
`filename`	specifica il nome del file da caricare come sistema operativo (deve essere riconosciuto come nome valido dal protocollo, in genere TFTP, usato dal client per caricarlo).
`next-server`	specifica il server da cui scaricare il file per il boot del sistema (indicato dalla precedente `filename`).

Tabella 8.6: Principali direttive del file `dhcpd.conf`.

In generale il DHCP consente di suddividere gli IP restituiti alle stazioni in diverse sottoreti, anche se nel nostro esempio ne viene usata una soltanto. Questo è fatto tramite la direttiva `subnet`, che serve a dichiarare una sottorete; essa è seguita dall'indirizzo della rete e dalla specificazione della relativa netmask preceduta dalla parola chiave `netmask`. All'interno della dichiarazione si potrà poi indicare l'uso di una assegnazione dinamica specificando l'intervallo di indirizzi con la direttiva `range`, oltre ai tempi di *lease* e alle altre opzioni DHCP da usare per quella sottorete.

Un'altra dichiarazione utile è quella che consente di configurare una stazione singola (analoga dell'assegnazione statica di BOOTP), anch'essa mostrata nell'esempio in sez. 8.2.3. Questo viene fatto tramite la direttiva `host` seguita dal nome della macchina, quest'ultima sarà identificata dal *MAC address* specificato dalla direttiva `hardware ethernet`, mentre l'indirizzo fisso da assegnare verrà specificato dalla direttiva `fixed-address`.

Si noti come in entrambi questi esempi di dichiarazioni si sono dovute poi specificare i valori dei parametri da applicare con una serie di altre direttive semplici poste all'interno di un blocco

Direttiva	Significato
subnet	dichiara una sottorete, ad essa deve seguire l'indirizzo IP della sottorete e la relativa maschera introdotta dalla parola chiave netmask; il corpo della dichiarazione conterrà i parametri da applicare alla suddetta sottorete.
host	dichiara una macchina singola, ad essa deve seguire il nome della stessa; il corpo della dichiarazione conterrà i parametri da applicare alla suddetta macchina.
group	dichiara un gruppo di macchine, ad essa deve seguire il nome del gruppo; il corpo della dichiarazione conterrà i parametri da applicare alle macchine del gruppo identificate attraverso ulteriori dichiarazioni di tipo host.

Tabella 8.7: Principali direttive di dichiarazione del file `dhcpd.conf`.

delimitato da parentesi graffe. Un elenco delle principali direttive di dichiarazione è illustrato in tab. 8.7. Per una lista completa dei parametri si può fare riferimento alla pagina di manuale del file di configurazione accessibile con `man dhcpd.conf`.

Una parte essenziale della configurazione del server è quella relativa alle opzioni DHCP, che costituiscono il contenuto delle risposte che il server fornisce ai client. È attraverso queste opzioni che si inviano ai client le varie informazioni sulla rete che ne permettono la configurazione automatica. Come già visto esse possono essere inserite sia direttamente nel corpo principale di `dhcpd.conf`, dove assumeranno il ruolo di valore di default, che all'interno di singole dichiarazioni per sotto-reti o macchine singole.

Opzione	Significato
domain-name	specifica il nome di dominio in cui ci si trova (ad uso dell'impostazione automatica di resolv.conf).
domain-name-servers	specifica una lista di server DNS (anche questo per l'impostazione automatica dei relativi campi in resolv.conf).
subnet-mask	specifica la netmask per la rete associata all'indirizzo assegnato.
broadcast-address	specifica l'indirizzo di *broadcast* per la rete associata all'indirizzo assegnato.
routers	specifica il *default gateway* per la rete associata all'indirizzo assegnato.
host-name	specifica il nome della macchina.
netbios-name-servers	specifica l'indirizzo di un server WINS (usato dalle macchine Windows).

Tabella 8.8: Principali opzioni del protocollo DHCP.

Le opzioni vengono sempre impostate da una direttiva nella forma `option` ***`nome-opzione valore-opzione`***`;` ed una lista dei nomi delle principali opzioni utilizzate per la configurazione automatica delle macchine è riportata in tab. 8.8. Al solito l'elenco completo è disponibile sulla relativa pagina di manuale, accessibile con `man dhcp-options`.

8.3 Amministrazione remota con SSH

Un tempo parlare di amministrazione remota di un computer significava parlare del comando `telnet`. Come accennato in sez. 7.7.1 però questo comando esegue tutte le sue comunicazioni in chiaro, password comprese, che così sono estremamente facili da intercettare. Per questo

oggi parlare di amministrazione remota significa parlare di SSH, sigla che sta per ***s**ecure **sh**ell*,[10] ed identifica un protocollo di comunicazione che permette di creare un canale cifrato con una macchina remota. Benché sia possibile utilizzare il canale in maniera generica, l'uso principale di SSH è quello di fornire una shell, consentendo così l'amministrazione remota; da questo origina il nome del servizio.

8.3.1 La configurazione del server SSH

Come buona parte dei protocolli di rete, SSH è realizzato con una architettura client-server; quando si vuole poter effettuare l'amministrazione da remoto di una macchina occorre installare su di essa la parte server del protocollo che viene fornita dal demone `sshd`. È questo demone che si incarica di rispondere alle richieste dei client e di realizzare tutte le fasi necessarie alla creazione di una canale cifrato su cui sia possibile inviare in maniera sicura le informazioni, per poi eseguire sulla macchina i vari comandi richiesti, restituendone i risultati.

Nel nostro caso faremo riferimento alla implementazione del protocollo realizzata dal progetto *OpenSSH*, il programma originale infatti, dopo essere stato rilasciato per un certo tempo con licenza libera, è diventato proprietario, ma un gruppo di programmatori indipendente[11] è riuscito, partendo dalla ultima versione libera disponibile, a ricreare una implementazione completa del protocollo, che nel frattempo è stato anche standardizzato.

Il demone può essere lanciato direttamente dalla linea di comando ed andrà automaticamente in background sganciandosi dal terminale. Il servizio ascolta di default sulla porta 22, ma può essere impostato su una porta diversa usando l'opzione `-p`. Con l'opzione `-D` lo si può eseguire in modalità di debug, mantenendolo agganciato al terminale, l'uso dell'opzione `-d`, ripetuta fino ad un massimo di tre volte, fa sì che esso generi messaggi di controllo sempre più dettagliati.

Nelle normali operazioni però, come gli altri demoni, anche `sshd` viene di norma lanciato automaticamente dagli script di avvio.[12] Il comportamento del demone (ad esempio anche la porta da utilizzare) è comunque controllato dal rispettivo file di configurazione; tutti i file di configurazione attinenti a SSH vengono normalmente mantenuti nella directory `/etc/ssh/`; di default la configurazione del server è posta nel file `sshd_config`, ma se ne può specificare un altro con l'opzione `-f`. Per un elenco completo delle opzioni di `sshd` si può fare riferimento alla pagina di manuale.

Come consuetudine il file di configurazione è in formato testo con una direttiva per linea, questa viene identificata da una parola chiave e seguita da uno o più argomenti separati da spazi o tabulazioni; righe vuote e tutto ciò che segue il carattere "`#`" al solito viene ignorato. Le principali direttive ed il relativo significato sono riportate in tab. 8.9; si sono descritte solo quelle che è più probabile che un amministratore si trovi a dover modificare. I valori delle altre sono

[10] il nome origina in realtà da un altro programma, `rsh`, parte dei cosiddetti `r-command` di Sun, che consentiva di eseguire una shell remota; gli `r-command`, in cui la "r" sta per *remote*, erano un insieme di *duplicati* dei comandi unix (`rsh`, `rcp`, `rlogin`) che eseguivano lo stesso compito degli analoghi senza la "`r`" su una macchina remota; il problema è che questo veniva fatto senza nessuna cifratura della comunicazione e consentendo l'esecuzione sulla base dell'indirizzo IP da cui i comandi venivano lanciati, in maniera cioè totalmente insicura.

[11] il progetto OpenSSH è portato avanti dagli stessi sviluppatori del sistema operativo OpenBSD, una versione di BSD il cui sviluppo è centrato sulla massima attenzione alle problematiche di sicurezza.

[12] su Debian lo script è `/etc/init.d/ssh`, altre distribuzioni usano `sshd`; si tenga presente che alcune distribuzioni recenti installano di default soltanto il client, il server viene fornito a parte (ad esempio su Debian a partire da *Etch* deve essere installato il pacchetto `openssh-server`).

di norma preimpostati a quanto specificato dal file di configurazione installato con il pacchetto: l'elenco completo con tutti i dettagli è nella pagina di manuale, accessibile con `man sshd_config`.

Direttiva	**Significato**
`AcceptEnv`	indica le variabili di ambiente inviate dal client che possono essere inserite nell'ambiente della sessione di lavoro.
`AllowTCPForwarding`	abilita l'inoltro di connessioni TCP sul canale cifrato (vedi sez/ 8.3.4).
`AllowAgentForwarding`	abilita l'inoltro delle connessioni all'agent (vedi sez. 8.3.3); prende come argomento i valori `yes` o `no`, di default è abilitato.
`AllowGroups`, `DenyGroup`	abilita/disabilita l'accesso agli utenti che fanno parte della lista di gruppi (specificata dai rispettivi nomi separati da spazi) passata come argomento; accetta anche i caratteri "?" e "*" come *wildcard* per i nomi dei gruppi.
`AllowUsers`, `DenyUsers`	abilita/disabilita l'accesso agli utenti che fanno parte della lista passata come argomento; accetta "?" e "*" come *wildcard*.
`Banner`	invia al client il contenuto del file passato come argomento per poterlo stampare come avviso prima di passare alla procedura di autenticazione, il default è `none` che non invia nulla.
`HostKey`	indica quali sono i file contenenti le chiavi crittografiche usate dal server.
`ListenAddress`	permette di indicare un indirizzo specifico su cui mettersi in ascolto (nella forma `indirizzo:porta`); il default è ascoltare su tutti gli indirizzi locali.
`PasswordAuthentication`	abilita la ordinaria autenticazione a password, prende i valori `yes` o `no` e di default è abilitata.
`PermitRootLogin`	permette il login diretto all'amministratore; prende come argomento i valori `yes` o `no`; le versioni più recenti supportano anche `without-password` che consente l'accesso solo con una autenticazione diversa dalle password.
`Port`	specifica la porta su cui mettersi in ascolto (di default è la 22).
`Protocol`	indica la versione del protocollo e deve essere impostato a 2 in quanto le precedenti versioni sono insicure; è possibile abilitare versioni inferiori solo a fine di compatibilità, ma si consiglia energicamente di aggiornare i client.[13]
`PubkeyAuthentication`	abilita l'autenticazione a chiavi (vedi sez. 8.3.3), prende i valori `yes` o `no` e di default è abilitata.
`Subsystem`	configura un sottoprogramma, e richiede come argomenti il nome dello stesso seguito dal pathname al programma che lo realizza; viene in genere utilizzato solo per abilitare SFTP, un protocollo di trasferimento file con una sintassi identica a quella di FTP, con il sottosistema denominato `sftp` ed il programma `sftp-server`.
`X11DisplayOffset`	assegna un valore di partenza per il numero di *display* (vedi sez. 3.3.4) usato dalle sessioni *X Window* inoltrate sul canale cifrato.
`X11Forwarding`	abilita l'*inoltro* delle sessioni X11 (vedi sez. 8.3.4); prende come argomento i valori `yes` o `no` e di default è disabilitato.

Tabella 8.9: Principali direttive di configurazione per il demone `sshd` usate nel file `sshd_config`.

Al solito per una descrizione completa delle varie direttive si può fare riferimento alla pagina di manuale, accessibile con `man sshd_config`. Un estratto del file di configurazione di `sshd` installato su una Debian Wheezy è riportato di seguito:

sshd_config
```
# What ports, IPs and protocols we listen for
Port 22
# Use these options to restrict which interfaces/protocols sshd will bind to
#ListenAddress ::
```

[13] che è il motivo per cui non descriveremo il formato di questa direttiva; chi vuole esporsi al rischio si legga attentamente le pagine di manuale.

```
#ListenAddress 0.0.0.0
Protocol 2
# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
#Privilege Separation is turned on for security
...
PermitRootLogin yes
StrictModes yes
RSAAuthentication yes
PubkeyAuthentication yes
#AuthorizedKeysFile       %h/.ssh/authorized_keys
# Don't read the user's ~/.rhosts and ~/.shosts files
IgnoreRhosts yes
# For this to work you will also need host keys in /etc/ssh_known_hosts
RhostsRSAAuthentication no
# similar for protocol version 2
HostbasedAuthentication no
# Uncomment if you don't trust ~/.ssh/known_hosts for RhostsRSAAuthentication
#IgnoreUserKnownHosts yes
# To enable empty passwords, change to yes (NOT RECOMMENDED)
PermitEmptyPasswords no
# Change to yes to enable challenge-response passwords (beware issues with
# some PAM modules and threads)
ChallengeResponseAuthentication no
# Change to no to disable tunnelled clear text passwords
#PasswordAuthentication yes
...
X11Forwarding yes
X11DisplayOffset 10
PrintMotd no
PrintLastLog yes
TCPKeepAlive yes
#UseLogin no
# Allow client to pass locale environment variables
AcceptEnv LANG LC_*
Subsystem sftp /usr/lib/openssh/sftp-server
UsePAM yes
```

Data la delicatezza delle operazioni svolte dal demone, tramite il quale diventa possibile eseguire un qualunque comando su una macchina remota, è stata posta la massima attenzione ai requisiti di sicurezza. Ad esempio di default l'accesso remoto all'utente di amministrazione è disabilitato, e può essere abilitato solo con la direttiva `PermitRootLogin` impostata al valore `yes`; senza di essa occorre prima collegarsi come un utente normale e poi usare `su`.

Si tenga presente che `sshd` onora sia la presenza del file `/etc/nologin` non consentendo, quando esso esiste, la connessione ad utenti che non siano l'amministratore, che l'utilizzo dei *TCP wrappers*, obbedendo alle restrizioni poste nei file `hosts.allow` e `hosts.deny` (vedi sez. 8.1.5). Inoltre con le direttive `AllowGroups`, `AllowUsers`, `DenyGroups`, `DenyUsers` si può restringere l'accesso ad utenti e gruppi specificati. Si tenga presente però che utilizzando sia `AllowUsers` che `AllowGroups` devono essere soddisfatte entrambe le condizioni, per cui se non c'è almeno uno degli utenti indicati che fa parte di uno dei gruppi indicati, nessuno potrà più collegarsi.

Se l'utente è fra quelli permessi il server ne eseguirà l'autenticazione; `sshd` supporta molti meccanismi di autenticazione, ciascuno dei quali può essere abilitato o meno in maniera indipendente tramite una opportuna direttiva nella forma *Meccanismo*`Authentication` (per l'elenco completo si rimanda alla pagina di manuale) impostata a `yes` o a `no`. Il default è la richiesta della password dell'utente, controllato dalla direttiva `PasswordAuthentication`; in tal caso normalmente `sshd` si appoggia a PAM, con tutte le funzionalità viste in sez. 4.3.7, ma una alternativa molto più valida (su cui torneremo in sez. 8.3.3) è l'autenticazione a chiavi, controllata dalla direttiva `PubkeyAuthentication`.

Infine una ulteriore serie di direttive (per cui di nuovo si rimanda alla pagina di manuale) consentono di controllare i dettagli della cifratura del canale (ad esempio gli algoritmi crittografici utilizzabili), SSH infatti usa la crittografia a chiave asimmetrica,[14] ed in particolare la direttiva `HostKey` consente di indicare quali chiavi devono essere utilizzate dal server. In genere queste sono mantenute nella directory `/etc/ssh/` insieme ai file di configurazione, in una serie di file `ssh_host_*_key` per le chiavi private e `ssh_host_*_key.pub` per le chiavi pubbliche, dove l'asterisco sta ad indicare il tipo di chiave (che può essere `dsa`, `ecdsa` o `rsa`).

Queste chiavi vengono usate dal server per essere identificato dai client e per generare le chiavi di sessione con cui sono cifrati le singole connessioni.[15] Questi file vengono generati automaticamente all'installazione del pacchetto (anche se è possibile generarli a mano, vedi sez. 8.3.3) ed in particolare quelli contenenti le chiavi private devono essere conservati con la massima cura in quanto costituiscono le credenziali che attestano l'identità del server.

Se l'autenticazione ha successo il server mette a disposizione del client un canale cifrato su cui eseguire una trasmissione sicura dei dati. L'uso più comune è quello di realizzare una sessione di lavoro in cui eseguire i programmi richiesti dal client (in genere una shell) facendo passare tutte le comunicazioni attraverso il canale cifrato. In tal caso la direttiva `AcceptEnv` indica quali fra le eventuali variabili d'ambiente richieste dal client potranno essere copiate da quest'ultimo alla sessione creata sul server.

Vedremo poi in sez. 8.3.4 come l'uso di SSH non si limiti all'esecuzione di comandi in remoto, ma consenta anche di eseguire l'*inoltro* sul canale cifrato di altre tipologie di dati. Ad esempio è possibile avere una sessione grafica remota, lanciando sul server programmi che usano l'interfaccia grafica di X11 ed ottenendo attraverso SSH le finestre sul client, per cui si potranno eseguire compiti di amministrazione remota anche attraverso programmi ad interfaccia grafica.

8.3.2 L'utilizzo di SSH come client

Prima di entrare nei dettagli dei vari comandi che utilizzano SSH dal lato client occorre precisare che anche questi sono controllati da un file di configurazione generale, `/etc/ssh/ssh_config`, la cui struttura è analoga a quella di `sshd_config`, con una direttiva per linea. In questo caso però la direttiva `Host` ha un ruolo speciale, essa infatti permette di indicare un insieme di macchine (identificate per indirizzo IP o nome a dominio) cui si applicano tutte le direttive seguenti fino ad una eventuale successiva direttiva `Host`.

---

[14] per una introduzione ai concetti su cui si basa la crittografia a chiave asimmetrica, e la relativa nomenclatura, si rimanda alla sezione 1.2 di [SGL].

[15] il protocollo prevede che ogni connessione sia cifrata in maniera indipendente con una *chiave di sessione*, rinnovata periodicamente, che viene creata tramite queste chiavi.

In sostanza una direttiva Host definisce una sezione del file contente le configurazioni da usare con tutte le macchine da essa definite. L'amministratore può così imporre delle restrizioni nell'accesso a certe macchine, ad esempio si può disabilitare di default la possibilità di far eseguire al client reinoltro delle sessioni X (vedi sez. 8.3.4) o richiedere specifici default per i parametri delle connessioni.

Direttiva	**Significato**
CheckHostIP	controlla se la macchina verso cui ci si collega è conosciuta (se cioè è presente nel file known_hosts) verificando se le chiavi pubbliche sono cambiate in modo da prevenire attacchi di tipo *man in the middle*; di default è impostata a yes.
Compression	specifica se comprimere il traffico, il valore può essere yes o no (il default).
CompressionLevel	se si è abilitata la compressione permette di specificarne il livello,[16]prende un valore numerico, ed il default è 6.
ForwardAgent	consente il reinoltro della connessione all'agent sulla macchina remota a cui ci si collega, in questo modo si può usare l'*agent* presente sulla propria macchina per ulteriori connessioni SSH partenti dalla macchina remota, senza dovere ripetere l'immissione di password; prende come argomento i valori yes o no.
ForwardX11	consente l'inoltro della sessione X (vedi sez. 8.3.4); prende come argomento i valori yes o no.
HashKnownHosts	richiede che i nomi delle macchine a cui ci si è già collegati salvati in known_hosts vengano salvati con un hash invece che in chiaro.
Host	restringe l'applicazione delle direttive seguenti ai collegamenti effettuati verso le macchine specificate; prende come argomento sia nomi a dominio che indirizzi IP, e supporta l'uso dei caratteri jolly "*" e "?"; l'uso del solo "*" permette di definire i valori di default da applicare a qualunque macchina remota.
Port	specifica la porta di destinazione da usare come default; è preimpostato a 22.
Protocol	indica le versioni del protocollo da utilizzare, deve essere sempre uguale a 2.
SendEnv	specifica quali variabili di ambiente presenti dal lato client devono essere passate al server, che deve comunque accettarle con la direttiva AcceptEnv.
ServerAliveCountMax	imposta il numero di messaggi di *keepalive* senza risposta dopo i quali viene abbattuta la connessione.
ServerAliveInterval	imposta il tempo (in secondi) senza traffico sulla connessione dopo il quale il client invia un messaggio di *keepalive* al server, oltre a rilevare le connessioni terminate, consente anche di mantenere attivi i collegamenti verso quelle macchine poste dietro firewall mal configurati che perdono le connessioni su cui non passa traffico per un certo intervallo di tempo.
StrictHostKeyChecking	definisce l'accuratezza con cui vengono controllate le macchine note, con yes known_hosts non viene mai aggiornato e le connessioni verso macchine con chiavi pubbliche cambiate sono bloccate; con ask (il default) le nuove macchine vengono aggiunte a known_hosts previa conferma esplicita da dare alla prima connessione, con no le connessioni verso macchine con chiavi pubbliche cambiate non vengono bloccate.

Tabella 8.10: Principali direttive di configurazione del file ssh_config per i client SSH.

Le principali direttive ed il relativo significato sono illustrate in tab. 8.10; al solito l'elenco completo, la spiegazione dettagliata delle stesse e i loro possibili valori sono riportati nella pagina di manuale, accessibile con man ssh_config. Gli utenti possono inserire le proprie configurazioni

[16]la compressione è effettuata tramite il programma gzip, ed il livello di compressione impostato da questa direttiva è quello usato da detto programma, che prevede la possibilità di scegliere una maggiore compressione a scapito di maggiore tempo per eseguirla, o viceversa; i valori vanno da 1 (veloce) a 9 (lenta); la scelta del valore ottimale dipende in maniera essenziale dalle caratteristiche della macchina e della rete, su una macchina lenta con connessione veloce una compressione alta può ridurre le prestazioni per il tempo perso ad eseguirla.

personali nel file `~/.ssh/config`, che, come tutte le configurazioni personali di SSH relative un utente (sia per il lato client che per quello server) vengono mantenute nella directory `.ssh` della sua home. Un esempio di `/etc/ssh/ssh_config` (estratto della versione installata da Debian Wheezy) è il seguente:

ssh_config
```
...
Host *
#   ForwardAgent no
#   ForwardX11 no
#   RhostsAuthentication no
#   RhostsRSAAuthentication no
#   RSAAuthentication yes
#   PasswordAuthentication yes
...
    SendEnv LANG LC_*
    HashKnownHosts yes
    GSSAPIAuthentication yes
    GSSAPIDelegateCredentials no
```

Si tenga presente che nell'uso dei comandi client le configurazioni applicate vengono determinate esaminando prima le opzioni date sulla riga di comando, poi (se presente) il file di configurazione personale in `~/.ssh/config` ed infine il file di configurazione generale. Il primo valore ottenuto è quello che viene usato, questo vuol dire che le opzioni a riga di comando hanno la precedenza sulle impostazioni personali, che a loro volta hanno la precedenza su quelle di `/etc/ssh/ssh_config`. Inoltre se ci sono più corrispondenze per diverse sezioni `Host` verrà ovviamente utilizzata la prima, questo significa che è opportuno mettere le configurazioni più specifiche in testa al file, e che qualunque altra sezione venga messa dopo una sezione `Host *` non avrà nessun effetto.

I due comandi principali di utilizzo di SSH dal lato client sono `ssh` e `scp` utilizzati rispettivamente per l'amministrazione remota e per la copia dei file. Il comando più importante resta comunque `ssh` anche se buona parte di quello che diremo nella trattazione seguente (in particolare quanto riguarda la creazione del canale cifrato e l'autenticazione) resta valido anche per tutti gli altri comandi che operano come client rispetto a `sshd`.

Se lanciato senza argomenti, `ssh` stampa un messaggio di aiuto, il comando infatti richiede che si specifichi almeno un argomento, che indica la macchina a cui collegarsi. Se presente il secondo argomento indica il programma da eseguire sulla macchina remota nella sessione di collegamento; in questo caso la connessione sarà automaticamente chiusa una volta conclusa l'esecuzione del comando indicato. Utilizzando un solo argomento sarà invece eseguita una shell.

Se non si specifica niente il collegamento con la macchina remota viene sempre eseguito per conto dell'utente che esegue `ssh`; se si desidera utilizzare un utente diverso se ne può specificare l'username con l'opzione `-l` o usare una indicazione del tipo `utente@macchina` per specificare il server a cui ci si vuole connettere. Alla fine la forma più semplice di invocazione del comando è:

```
ssh [-l nome_login] nome_macchina | nome_login@nome_macchina [comando]
```

Una volta messo in esecuzione per prima cosa `ssh` si collega al server per creare il canale cifrato. Il protocollo prevede che il server invii la sua chiave pubblica che verrà confrontata con quella presente nell'elenco dei server noti. Questo viene mantenuto nel file `known_host` della

directory `.ssh` dell'utente; se il server a cui ci si rivolge è già presente in `known_host` verrà verificato che la chiave pubblica sia uguale a quella ivi registrata, comportamento che si ottiene dalla configurazione di default dalle direttive `CheckHostIP` e `StrictHostKeyChecking`, e se questo avviene si passa alla fase di autenticazione. Altrimenti la connessione verrà rifiutata con un avviso in cui viene riportata la riga di `known_host` che contiene il valore non corrispondente.

Se invece la macchina a cui ci si collega non è presente nell'elenco di `known_host` verrà stampata la fingerprint della sua chiave pubblica, così che la si possa verificare in maniera indipendente, e verrà chiesto se aggiungerla all'elenco. Solo una volta che si sia data conferma dell'aggiunta il procedimento passerà alla fase di autenticazione. Questa procedura è essenziale per accorgersi di eventuali attacchi di interposizione (i cosiddetti *man in the middle*) in cui un malintenzionato si spaccia per il server a cui ci si vuole collegare, in modo da poter intercettare il traffico verso di esso.

Una volta riconosciuta opportunamente la macchina remota il server viene creato un canale di comunicazione cifrato, e se previsto il server invia al client il contenuto del file specificato dalla direttiva `Banner` di tab. 8.9, che consente di replicare sulle sessioni SSH il funzionamento del file `/etc/issue`; poi viene eseguita la fase di autenticazione. Questa, come già detto in sez. 8.3.1, prevede diversi meccanismi (ne vedremo uno specifico in sez. 8.3.3), ma normalmente consiste semplicemente nel replicare la procedura di login che si fa sulla console con la richiesta della password dell'utente con cui ci si è collegati; sarà compito di `sshd` autenticare l'utente sul server usando l'username avuto dal client.

Una volta completata l'autenticazione il server provvederà a collegare il canale cifrato ad un terminale virtuale sul quale saranno eseguiti i comandi (con i privilegi dell'utente con cui ci si è connessi), connettendolo al terminale su cui è eseguito `ssh`, così che da esso si possano inviare dati ed ottenere risposte. Questo avviene soltanto quando non si specifica l'esecuzione di un comando e deve essere allocato un terminale ad uso della shell; se si richiede un singolo comando questo viene semplicemente eseguito con lo standard input e lo standard output collegati al canale cifrato, ed in tal caso per forzare l'allocazione di un terminale (necessario per certi comandi) occorre usare l'opzione `-t`.

Nella modalità standard in cui si usa `ssh` per invocare una shell remota il server eseguirà anche delle inizializzazioni specifiche;[17] in particolare prima di lanciare la shell verranno eseguite le istruzioni presenti nel file `/etc/ssh/sshrc` o, se questo esiste, nel file `.ssh/rc` dell'utente (quello della macchina remota, non quello da cui si usa `ssh`). In questo modo è possibile effettuare delle impostazioni specifiche (ad esempio attivare dei limiti) tramite questi file che sono gli analoghi di `/etc/profile` per le connessioni SSH.

Una delle opzioni principali di `ssh` è `-X`, che, se il server lo consente, attiva l'inoltro della sessione X11 (vedi sez. 8.3.4); questo può essere disabilitato (qualora abilitato per default in `ssh_config`) con l'opzione `-x`. Analogamente si può abilitare o disabilitare l'inoltro verso l'agente di autenticazione (che tratteremo in sez. 8.3.3) rispettivamente con le opzioni `-A` e `-a`. Nel caso il server sia stato posto in ascolto su una porta diversa da quella standard se ne può specificare esplicitamente una con l'opzione `-p`. Le altre opzioni più comuni sono illustrate in tab. 8.11, per un elenco completo e tutti i dettagli si rimanda al solito alla pagina di manuale del comando che è accessibile con `man ssh`.

[17] per la descrizione completa delle operazioni compiute dal server prima di arrivare al prompt della riga di comando remota si legga la pagina di manuale di `sshd`.

Opzione	Significato
-A	abilita l'inoltro della connessione all'agente di autenticazione.
-a	disabilita l'inoltro della connessione all'agente di autenticazione.
-i	specifica la chiave da usare per l'autenticazione (vedi sez. 8.3.3).
-L	abilita il *port forwarding* locale (vedi sez. 8.3.4).
-l	specifica l'utente con cui collegarsi al server.
-p	specifica la porta a cui collegarsi sul server.
-R	abilita il *port forwarding* remoto (vedi sez. 8.3.4).
-t	forza l'allocazione di un terminale virtuale, così da consentire anche l'esecuzione di comandi che ne richiedono la presenza.
-v	attiva la modalità di controllo in cui vengono stampati avvisi progressivi nella fase di connessione, può essere invocata fino a tre volte per aumentare la *prolissità*.
-X	attiva l'inoltro della sessione X11 (vedi sez. 8.3.4).
-x	disattiva l'inoltro della sessione X11 (vedi sez. 8.3.4).

Tabella 8.11: Principali opzioni del comando `ssh`.

L'altro comando client più utilizzato è `scp` che permette di copiare file via rete in maniera sicura, trasmettendoli attraverso un canale cifrato con SSH. La sintassi è analoga a quella del comando `cp`, e prevede l'indicazione di uno o più file (o directory) di origine da copiare su una destinazione. Solo che in questo caso l'indicazione può essere effettuata con una espressione generalizzata che consente di indicare anche una origine o una destinazione remota,[18] da scrivere nella forma:

```
utente@macchina:/percorso/al/file
```

dove `macchina` indica l'indirizzo (numerico o simbolico) della macchina remota su cui si trova il file, `utente` è l'username con il quale ci si vuole collegare alla macchina remota (lo si può omettere se corrisponde a quello con cui si sta lavorando), e `/percorso/al/file` il pathname del file sulla macchina remota; se quest'ultimo viene specificato in forma relativa viene considerato a partire dalla home dell'utente che si sta usando.

Un esempio di uso del comando potrebbe essere il seguente:

```
scp piccardi@gapil.gnulinux.it:gapil.pdf .
```

che copia il file `gapil.pdf` dalla home dell'utente su un server remoto alla directory corrente, mentre per effettuare un trasferimento in direzione opposta di una intera directory si potrà usare il comando:

```
scp -r gapil piccardi@gapil.gnulinux.it:public_html/
```

che copia ricorsivamente il contenuto della directory `gapil` nella directory `public_html` della home del server.

Il comando richiede la stessa procedura di autenticazione usata da `ssh`, che normalmente comporta l'immissione da terminale delle password relative agli utenti delle macchine a cui ci

[18] strettamente o l'origine o la destinazione, `scp` crea una connessione SSH verso una singola macchina e quindi non può supportare la copia da una macchina remota ad un'altra.

si collega. L'opzione `-p` permette di preservare i tempi ed i permessi del file originale, mentre l'opzione `-r` esegue una copia ricorsiva di intere directory. Le altre principali opzioni sono in tab. 8.12, al solito per i dettagli su tutte le altre opzioni si può fare riferimento alla pagina di manuale accessibile con `man scp`.

Opzione	Significato
`-C`	abilita la compressione della connessione.
`-i`	specifica la chiave da usare per l'autenticazione (vedi sez. 8.3.3).
`-l`	imposta una limitazione nell'uso della banda, da esprimersi in Kbit/s.
`-P`	specifica la porta a cui collegarsi sul server.
`-p`	preserva tempi e permessi dei file che si copiano.
`-r`	esegue una copia ricorsiva delle directory.
`-v`	attiva la modalità di controllo in cui vengono stampati avvisi progressivi nella fase di connessione, può essere invocata fino a tre volte per aumentare la *prolissità*.

Tabella 8.12: Principali opzioni del comando `scp`.

Infine per effettuare trasferimenti di file è disponibile come alternativa a `scp` anche una interfaccia che riproduce quella del protocollo FTP, utilizzabile attraverso il comando `sftp`. Nell'uso più comune questo comando prende come argomento l'indirizzo di una macchina remota a cui collegarsi e crea una sessione di lavoro da cui si possono inviare i comandi di tab. 7.37, con sintassi del tutto analoga a quella di un client FTP, per cui sono disponibili operazioni più complesse del semplice trasferimento di un file (ad esempio ottenere una lista dei file presenti in remoto). Resta comunque possibile trasferire i file in maniera immediata usando la sintassi di `scp`. Inoltre si può specificare con l'opzione `-b` un *batch file* da cui prendere i comandi per una esecuzione non interattiva.

Un primo vantaggio di `sftp` è che tutto quello che occorre sul lato server è una installazione di `sshd` che abbia abilitato il supporto relativo (questo viene fatto di default in tutte le installazioni standard) e non è necessario installare un demone a parte. Inoltre tutti i vantaggi di SSH restano a disposizione, dalla cifratura dei dati alla possibilità di avere metodi di autenticazione avanzata come quello che vedremo in sez. 8.3.3.

Se a questo si aggiunge il fatto che questa interfaccia è stata standardizzata con il nome di SFTP, ed è disponibile come estensione per molti client FTP e programmi che girano anche sotto Windows come Filezilla e WinSCP, si può tranquillamente affermare che al giorno d'oggi non esistono più ragioni valide per installare un server FTP.

8.3.3 La autenticazione a chiavi

Una delle caratteristiche più importanti di SSH è quella di avere, rispetto alla classica autenticazione con username e password, la possibilità di utilizzare diverse modalità di autenticazione e fra queste la più interessante è quella basata sulle chiavi crittografiche. Questa modalità di autenticazione permette, caso quasi unico nel suo genere, di aumentare la sicurezza ottenendo anche una maggiore facilità d'uso, ad esempio diventa possibile avere delle sessioni di lavoro in cui cui una volta effettuato un accesso iniziale non è più necessario ripetere l'immissione di una password di autenticazione tutte le volte che si utilizza uno dei comandi `ssh` o `scp`.

Questa modalità di autenticazione si basa sulla crittografia a chiave asimmetrica alla base del protocollo, sfruttando l'uso di una coppia di chiavi personali dell'utente. Una delle caratteristiche della crittografia a chiave asimmetrica infatti è che è possibile effettuare il riconoscimento univoco di un utente, una volta che questo dimostri di essere in possesso della chiave privata associata alla chiave pubblica che il server utilizza per identificarlo (di nuovo per una trattazione più dettagliata dei concetti relativi alla crittografia a chiave asimmetrica si rimanda la sezione 1.2 di [SGL]).

Nel caso di autenticazione a chiave, invece di mandare la password dell'utente sul canale cifrato per farla verificare, viene usata una procedura di *challenge and response*: il server invia al client un segreto (la *sfida*) cifrato con la chiave pubblica dell'utente a cui si vuole garantire l'accesso, solo il reinvio del segreto decifrato (la *risposta*) è garanzia che il client conosce la chiave privata, ed è questo che garantisce l'identità dell'utente.

Questa modalità di autenticazione è molto più sicura di quella basata su username e password perché anche se il server venisse compromesso non riceverebbe comunque le credenziali personali di un utente: nel primo caso otterrebbe la password, qui invece la chiave privata resta sempre sul client. Inoltre in questo modo si possono rendere del tutto inutili gli attacchi a dizionario (di cui abbiamo già parlato in sez. 4.3.2) anche nel caso in cui un utente avesse scelto una password debole, dato che l'uso delle password può essere completamente escluso.

Per poter utilizzare questa modalità di autenticazione occorre anzitutto generare una coppia di chiavi; il pacchetto *OpenSSH* mette a disposizione un apposito programma per la creazione e la gestione delle chiavi, `ssh-keygen`. Se invocato senza nessuna opzione `ssh-keygen` predispone la creazione di una chiave RSA, chiedendo su quale file salvarla e proponendo il default di `id_rsa` nella solita directory `.ssh` nella home dell'utente. Viene poi richiesta una *passphrase*, da indicare due volte, che serve a proteggere l'accesso alla chiave privata, che non può essere utilizzata senza di essa.

Si può indicare un diverso formato per la chiave con l'opzione `-t` seguita da un parametro che indica il tipo di chiave i cui possibili valori sono tre: `rsa` per chiavi RSA di default, `dsa` per chiavi DSA e `ecdsa` per le chiavi ECDSA.[19] Le chiavi private vengono salvate, a meno di non indicare in fase di creazione un file diverso, sempre sotto `.ssh` nella home dell'utente rispettivamente nei file `id_rsa`, `id_dsa` e `id_ecdsa`, le corrispondenti chiavi pubbliche sono salvate sempre su un file con lo stesso nome della chiave privata esteso con un `.pub`. Si può inoltre specificare il nome direttamente sulla riga di comando, usandolo come parametro per l'opzione `-f`.

Una volta completato il processo di generazione, che con macchine lente e per chiavi di grandi dimensioni può essere anche molto lungo, il comando salva la chiave privata cifrata con la *passphrase* fornita. Se si vuole cambiare la *passphrase* in un secondo tempo si può chiamare il comando con l'opzione `-p`: verrà richiesta la vecchia *passphrase* per accedere alla chiave privata, e poi la nuova, che andrà ripetuta due volte per conferma. Alle chiavi è pure associato un commento, che non ha nessun uso nel protocollo, ma serve da identificativo della chiave; esso viene di solito inizializzato alla stringa `utente@hostname` e può essere cambiato con l'opzione `-c`.

Infine con l'opzione `-l` si può stampare l'*impronta digitale* (la cosiddetta *fingerprint*) della chiave a scopo di verifica, mentre l'opzione `-b` permette di specificare la lunghezza della chiave in bit. Il valore di default, 1024, è in genere adeguato, lunghezze maggiori aumentano la sicurezza,

[19] in realtà è possibile usare anche `rsa1` per ottenere il formato di chiavi RSA usato nella vecchia versione protocollo, ma queste sono comunque da evitare per l'insicurezza dello stesso.

ma richiedono molto più tempo per la generazione e l'uso. I dettagli sul funzionamento del comando e le restanti opzioni sono accessibili nella pagina di manuale con `man ssh-keygen`.

Una volta creata la coppia di chiavi diventa possibile utilizzarla per l'autenticazione. Il meccanismo prevede che quando un utente invoca `ssh`, se c'è una chiave nella directory `.ssh` (nella home dell'utente che effettua il collegamento, quindi dalla parte del client) la parte pubblica venga inviata al server. Quando `sshd` riceve una richiesta di connessione in questa modalità va a verificare la presenza del file `.ssh/authorized_keys` nella home dell'utente per conto del quale si vuole accedere (cioè dalla parte del server, dove l'utente può essere anche completamente diverso) e se in esso è presente la chiave pubblica ricevuta viene eseguita la procedura di *challenge and response* illustrata in precedenza; a questo punto `ssh` richiederà la *passphrase* per sbloccare la chiave privata e rispondere alla sfida.

Per quanto appena detto quello che si deve fare per abilitare l'accesso con l'autenticazione a chiavi è inserire sulla macchina remota la chiave pubblica della persona a cui si vuole dare l'accesso. Questa deve essere aggiunta al file `authorized_keys` nella directory `.ssh` dell'utente per conto del quale si vuole dare accesso. Se si è trasferita la chiave pubblica sul server basterà un comando del tipo:

```
cat id_dsa.pub >> .ssh/authorized_keys
```

dove nel caso `id_dsa.pub` è la chiave pubblica dell'utente che deve poter accedere.

Si noti l'uso della redirezione in *append*, il file `authorized_keys` infatti può contenere più chiavi pubbliche, le chiavi infatti sono mantenute una per riga in formato codificato ASCII, per cui le singole righe del file sono di norma molto lunghe. Se si fosse semplicemente sovrascritto il file si sarebbero cancellate quelle di eventuali altri utenti.

Si noti anche che non è necessario che chi accede con l'autenticazione a chiavi conosca la password dell'utente sulla macchina remota, basta che la sua chiave sia fra quelle autorizzate. In questo modo diventa possibile dare accesso anche a più persone senza che queste debbano condividere la conoscenza di una unica password, cosa che ne aumenterebbe il rischio di scoperta o perdita. In questo caso infatti l'autenticazione viene fatta solo sulla base della corrispondenza fra una chiave pubblica ed una chiave privata e tutto quello che serve è la presenza della coppia di chiavi. Ancora una volta si è ottenuta maggior sicurezza ed una semplificazione nell'uso non essendo più necessario ricordarsi una password diversa per ciascun server a cui ci si deve collegare, ma solo quella della propria chiave.

Dato che il procedimento manuale di inserimento della chiave può essere macchinoso (richiede la copia del file della chiave pubblica, e poi il collegamento sulla macchina per inserirla) con il pacchetto di SSH viene fornito un apposito comando, `ssh-copy-id`, che permette di automatizzare l'operazione aggiungendo la chiave pubblica dell'utente che lo invoca all'`authorized_keys` di una macchina di destinazione. Un esempio di uso del comando è il seguente:

```
ssh-copy-id piccardi@oppish.truelite.it
```

dato il quale si dovrà immettere la password dell'utente remoto.

Le vecchie versioni del comando richiedono l'uso esplicito dell'opzione `-i` per indicare quale è il file della chiave da usare, si tenga conto che in questo caso va indicato il file della chiave privata (a trovare quello della chiave pubblica ci pensa il comando). Nelle versioni recenti, se si sta utilizzando l'agente di autenticazione `ssh-agent` (che tratteremo a breve), la chiave viene presa direttamente da lui.

Ovviamente quando si usa questo metodo di autenticazione la protezione della propria chiave privata è fondamentale, chiunque ne venga in possesso e possa utilizzarla ottiene immediatamente tutti gli accessi a cui essa è abilitata, e questo è il motivo per cui essa viene protetta con una *passphrase*[20] che deve essere fornita tutte le volte che la si deve usare. Essa comunque deve essere protetta sia da lettura che da scrittura da parte di estranei. Questo è imposto dal comando `ssh` stesso, che si rifiuta di usare, stampando un clamoroso avvertimento, una chiave privata che sia leggibile da altri oltre al proprietario.

A questo punto però ci si potrebbe chiedere l'utilità di tutto questo armamentario, dato che invece di una password per il login bisogna comunque inserire una *passphrase* per sbloccare la chiave privata, che si richiede tra l'altro essere lunga e difficile. L'utilità sta nel fatto che è possibile usare un altro programma, l'agente di autenticazione `ssh-agent`, che permette di sbloccare la chiave una sola volta, all'inizio di una sessione di lavoro, e poi mantiene la chiave privata sbloccata in memoria, permettendone il successivo riutilizzo senza che sia necessario fornire di nuovo la *passphrase*.

L'idea è che `ssh-agent` venga lanciato all'inizio di una sessione di lavoro, in modo che tutti i comandi lanciati successivamente possano farvi riferimento, interrogandolo tutte le volte che è necessario compiere una operazione con la propria chiave privata. Il programma svolge il ruolo di intermediario, per questo si parla di agente: infatti è sempre `ssh-agent` ad eseguire le operazioni di autenticazione, fornendo ai richiedenti i risultati ottenuti, in modo da non dover mai comunicare verso l'esterno la chiave privata.

Di norma `ssh-agent` viene lanciato automaticamente all'avvio delle sessioni grafiche, mentre lo si deve lanciare esplicitamente nel caso di collegamento da terminale. Si tenga presente che all'avvio l'agente non dispone di nessuna chiave privata e perché possa funzionare occorre consegnargli la propria chiave. Questo viene fatto con il comando `ssh-add`,[21] che chiederà la *passphrase* della chiave privata, la sbloccherà, e la invierà all'agente, che da quel momento in poi potrà utilizzarla. Sempre con `ssh-add` si potrà rimuovere la propria chiave dall'agente con l'opzione `-d`, o richiedere la rimozione di tutte le chiavi presenti con `-D`. Il comando consente inoltre di verificare le chiavi in possesso dell'agente con l'opzione `-l` (per le altre opzioni si consulti la pagina di manuale).

Per poter comunicare con l'agente i vari programmi che ne hanno bisogno usano la variabile di ambiente `SSH_AUTH_SOCK` che identifica il socket locale su cui esso è in ascolto. Se non è definita anche se il programma è attivo l'autenticazione a chiavi non funzionerà, e si passerà direttamente all'autenticazione normale. La variabile viene definita dallo stesso `ssh-agent` quando viene lanciato, che la scrive anche sullo standard output. Si ricordi però che le variabili di ambiente definite da un processo vengono viste soltanto nei processi figli, pertanto quando si lancia `ssh-agent` in *console*, questo resterà attivo in background, ma fintanto che non si inserisce manualmente la variabile `SSH_AUTH_SOCK` nel proprio ambiente, non potrà essere utilizzato.

Una funzionalità ancora più interessante dell'uso di `ssh-agent` è che `ssh` supporta anche il reinoltro della connessione che ha verso di lui. Se il server `sshd` a cui ci si collega consente di usare questa funzionalità (il default è di farlo) una volta che ci si è collegati ad esso la

---

[20] la *passphrase* viene a sostituire il ruolo della password, ed è bene sceglierla con molta cura, in quanto è l'unica protezione contro l'abuso della nostra chiave privata; per questo motivo si parla di *passphrase* e non di password; è quindi il caso di usare una frase lunga e complicata, e non una singola parola.

[21] in caso di sessioni grafiche esistono vari programmi alternativi, come `ssh-askpass`, che possono essere utilizzati per eseguire lo stesso compito.

variabile SSH_AUTH_SOCK viene automaticamente definita, ed sshd reinvierà all'indietro le richieste di autenticazione fatte sul server remoto. Il risultato è che si potrà usare ssh con l'autenticazione a chiavi anche sul server remoto, usando le chiavi che si stanno gestendo con il proprio agente di autenticazione locale senza che queste abbandonino la propria macchina.

Per utilizzare questa funzionalità occorre usare l'opzione -a (o inserire nel proprio .ssh/config la direttiva ForwardAgent yes). Il risultato è la capacità di usare ssh in maniera sicura con l'autenticazione a chiavi senza dover reimmettere le password non soltanto quando lo si esegue sulla propria macchina, ma anche quando lo si esegue sulle macchine su cui ci si è collegati a partire dalla propria (il salto però funziona una sola volta).

Questa funzionalità non viene abilitata di default perché espone al rischio che l'amministratore del server remoto possa utilizzare la connessione reinoltrata per sfruttare l'agente e, fintanto che si resta collegati, autenticarsi verso tutte le altre macchine per cui la propria chiave è abilitata. Ovviamente questo è un rischio significativo, si tenga presente però che se non ci si può fidare dell'amministratore di un server remoto, usarlo per connessioni verso altre macchine non è comunque una buona idea in generale visto che qualunque cosa facciamo può essere intercettata, comprese le password inserite in una normale chiamata a ssh. In questo caso l'unica soluzione sicura è di non collegarsi mai ad una macchina remota attraverso un'altra.

8.3.4 Le funzionalità di reinoltro del traffico

Una delle caratteristiche più interessanti di SSH, già abbondantemente citata, è la capacità di utilizzare la presenza di un canale cifrato per mettere a disposizione delle funzionalità di comunicazione che vanno ben al di là della semplice realizzazione di una sessione di lavoro remota.

La prima di queste funzionalità è quella dell'inoltro delle sessioni *X Window*, che consente di eseguire da remoto anche applicazioni con interfaccia grafica. La trasparenza sulla rete del protocollo di *X Window* già consentiva questo anche con una sessione fatta con telnet. Come spiegato in sez. 3.3.4 infatti basta definire sulla macchina remota la variabile d'ambiente DISPLAY con l'indirizzo IP della macchina su cui si vogliono ricevere le finestre, ed abilitare su di essa l'accesso al server X via rete con il comando xhost, per avere anche la trasmissione remota delle finestre.

Il problema resta quello che *X Window* non prevede nessuna cifratura dei dati, per cui una volta che si sia consentito un accesso da remoto al server X gli si potrà far fare di tutto, ed ovviamente questo è un grosso problema per la sicurezza. Ad esempio è relativamente semplice (ed è stato fatto) scrivere un programma che crea una finestra trasparente a tutto schermo messa in primo piano il cui solo scopo è quello di intercettare tutti i tasti premuti, ottenendo così un perfetto *keylogger* in grado di leggere tutte le password che vengono immesse, anche al di fuori della sessione X remota.

Con SSH invece si può ottenere l'inoltro della sessione X in maniera sicura, facendo transitare tutto il traffico sul canale cifrato, e senza la necessità di consentire l'accesso da remoto al server X, cosa che tra l'altro sulle distribuzioni più attente alle questioni di sicurezza, è disabilitata di default, per cui anche volendo non sarebbe possibile farlo, a meno di non cambiare la configurazione del server X e riavviarlo.

Dal lato server si può attivare l'inoltro della sessione X11 attraverso l'uso delle direttive X11Forwarding e X11DisplayOffset di sshd_config. La prima direttiva abilita l'inoltro della ses-

sione X11; di default questo è disabilitato (anche se molte distribuzioni installano un `sshd_config` che lo riabilita), in quanto si tratta di una funzionalità aggiuntiva che è il caso di usare solo se necessario: se ad esempio la macchina remota è un server sul quale non devono girare programmi con interfaccia grafica è assolutamente inutile (e quindi dannoso dal punto di vista della sicurezza) attivare questa funzionalità.

Nell'esempio di configurazione mostrato in precedenza si è attivata esplicitamente questa opzione. La seconda direttiva, `X11DisplayOffset`, consente di impostare il valore minimo del *display* (si ricordi quanto detto in sez. 3.3.3) usato come destinazione per la sessione X11. Questo in genere viene impostato al valore di 10, così da non creare conflitti con altri *display* eventualmente presenti in locale.

Quando si eseguono queste impostazioni quello che accade è che per ogni sessione SSH sulla macchina remota viene automaticamente definita la variabile `DISPLAY` al valore `localhost:NN` (con `NN` a partire da 10), così che le applicazioni lanciate nella sessione inviino i dati del protocollo X11 su un socket[22] su cui si pone in ascolto `sshd`. A questo punto il server trasferirà questi dati attraverso il canale cifrato al client, e quest'ultimo si incaricherà di inviarli al server X locale che disegnerà le finestre dell'applicazione remota.

Un'altra funzionalità avanzata messa a disposizione da `ssh` è quella del cosiddetto *reinoltro delle porte* (il *port forwarding*), che consente di far passare sul canale cifrato delle connessioni TCP generiche. Questa modalità può essere utilizzata in due modi, ma il più comune è quello in cui si inoltra tutto il traffico diretto ad una porta locale verso una qualunque porta posta su una qualunque macchina raggiungibile dalla macchina remota a cui ci si collega con SSH.

Questa prima modalità, in cui si inoltra il traffico a partire dalla macchina locale, viene attivata lanciando il comando `ssh` con l'opzione `-L` a cui deve seguire una terna indicante rispettivamente la porta locale su cui porsi in ascolto,[23] l'indirizzo IP della macchina remota e la porta remota cui verrà inoltrato il traffico ricevuto sulla porta locale. Queste informazioni devono essere separate dal carattere "`:`" e si possono usare sia valori numerici che simbolici; un esempio di utilizzo di questa opzione è il seguente:

```
ssh -L 1631:localhost:631 piccardi@davis.truelite.it
```

Nell'esempio si è collegato il traffico diretto verso la porta 1631 del *localhost* della macchina locale alla porta 631 del *localhost* della macchina remota (il `localhost` del comando è risolto sulla macchina remota). Questo consentirà di accedere sulla macchina locale all'interfaccia di configurazione di CUPS della macchina remota, nonostante questa sia posta in ascolto soltanto sul *localhost* di quest'ultima. Dato questo comando basterà puntare un browser sulla macchina locale alla porta 1631 del proprio *localhost* per ottenere la connessione alle pagine di CUPS della macchina remota. Si noti inoltre che come si è raggiunto `localhost`, si potrà anche raggiungere qualunque altro indirizzo visibile dalla macchina remota, ad esempio tutti quelli di una rete privata su cui essa si affacci.

La seconda modalità del *port forwarding* è quella in cui si inoltra il traffico a partire dalla macchina remota, che si attiva con l'opzione `-R`. In questo caso si può inoltrare tutto il traffico

[22] viene usato un socket messo in ascolto su `localhost` sulla porta `6000+NN` dato che le porte standard usate dalle sessioni di rete del protocollo X11 sono date da 6000 più il numero di display.

[23] in questa forma la porta locale sarà messa in ascolto soltanto su *localhost*, se si vuole utilizzare un altro indirizzo locale, lo si può specificare aggiungendolo prima della porta locale, seguito dal carattere di separazione ":" e dagli altri tre valori, ma questa forma è poco utilizzata.

diretto ad una porta della macchina remota verso una porta di una macchina raggiungibile dalla macchina locale. Come prima all'opzione deve seguire una terna analoga in cui si indicano nell'ordine: la porta su cui porsi in ascolto sulla macchina remota, l'indirizzo IP e la porta della macchina raggiungibile localmente a cui inoltrare il traffico inviato alla porta remota. Un esempio di questa modalità potrebbe essere il seguente:

```
ssh -R 20022:localhost:22 servizio@holland.truelite.it
```

In questo caso si consente a chi lavora sulla macchina remota `holland.truelite.it` di accedere in SSH, collegandosi localmente alla porta 20022, alla macchina da cui è stato lanciato questo comando. Diventa cioè possibile in generale consentire una connessione partente dalla macchina a cui ci si collega verso le macchine che stanno nella rete della macchina da cui si lancia il comando, anche se queste sono su una rete privata e non sono direttamente raggiungibili da Internet. Si tratta di una funzionalità che risulta molto comoda quando si devono raggiungere macchine dietro un firewall che possono soltanto uscire su Internet.

Si tenga presente che in entrambe le modalità quando ci si vuole mettere in ascolto su una porta privilegiata, occorre rispettivamente che l'utente che esegue il comando (nel caso di `-L`) o quello per conto del quale ci si collega (nel caso di `-R`) abbia i privilegi di amministratore, altrimenti la connessione SSH riuscirà ma senza il reinoltro del traffico.

8.4 La condivisione dei file sulla rete

Tratteremo in questa sezione i protocolli per la condivisione dei file via rete, concentrandoci in particolare sul protocollo più usato nei sistemi unix-like, cioè NFS. Tratteremo comunque, anche se in maniera estremamente superficiale, anche la condivisione dei file con i protocolli utilizzati in ambiente Windows attraverso l'utilizzo di Samba.

8.4.1 Il protocollo ed il server NFS

Il protocollo NFS, sigla che sta a significare *Network File System*, è stato creato da Sun per permettere di montare il contenuto di dischi che stanno su stazioni remote come se fossero presenti sulla macchina locale. In questo modo si può assicurare un accesso trasparente ai file, e si può utilizzare un filesystem anche su una macchina senza dischi,[24] passando solo attraverso la rete.

In genere tutte le distribuzioni prevedono i pacchetti per l'installazione di un server NFS. La sola scelta che si può dover fare è fra l'uso del supporto del protocollo direttamente a livello del kernel o l'uso di una implementazione realizzata completamente in user-space. Date le peggiori prestazioni, quest'ultima soluzione in genere viene evitata, e per questo in genere si deve utilizzare un kernel per cui sia stato abilitato il supporto per NFS, cosa che accade per i kernel standard installati da qualunque distribuzione.[25]

Il protocollo NFS è piuttosto complesso, nasce come protocollo *stateless* in cui ogni richiesta e successiva risposta è indipendente dalle altre e non esiste nessuna sessione stabilita fra client

[24] in questo caso la configurazione è più complessa, in quanto occorre abilitare il kernel stesso a montare la directory radice su NFS.

[25] le opzioni necessarie in caso di compilazione in proprio sono accessibili con `make menuconfig` sotto la voce `File system`, nella sotto voce `Network File System`.

e server. Il protocollo usava inizialmente soltanto pacchetti UDP (che si adattano bene al modello di richiesta/risposta indipendenti) ed è basato sul sistema delle RPC di cui abbiamo fatto una panoramica in sez. 7.6.5. Questo significa che al funzionamento di NFS, oltre a `portmap`, concorrono ben altri cinque demoni:

`rpc.nfsd` è il demone che svolge la gran parte del lavoro, realizzando le funzioni di accesso al contenuto dei file. Deve essere lanciato dopo che è stato attivato `portmap`.

`rpc.statd` è il demone che gestisce le caratteristiche dei file come tempi di accesso, permessi, ecc. Deve essere lanciato dopo che è stato attivato `portmap`.

`rpc.lockd` è il demone che gestisce, quando questo è abilitato, il *file locking*.[26] Di norma viene lanciato da `rpc.nfsd` in caso di necessità.

`rpc.mountd` è il demone che gestisce le richieste di montaggio del filesystem. Deve essere lanciato dopo che sono stati attivati `rpc.nfsd` e `rpc.statd`.

`rpc.rquotad` è il demone che permette, quando sono attivate, di gestire le quote sul filesystem di rete. Deve essere lanciato dopo che sono stati attivati `rpc.nfsd` e `rpc.statd`.

Del protocollo esistono diverse versioni, conseguenza dell'evoluzione subita nel tempo. La prima versione utilizzata è stata NFSv2 in quanto NFSv1 non è sostanzialmente mai uscita dai laboratori della SUN, questa presenta numerose limitazioni (fra cui una dimensione massima di 2Gb per i file) che sono state superate con NFSv3. Con la versione 4 sono state introdotte funzionalità di autenticazione forte, ed il protocollo è diventato *stateful*.

Benché complicato da descrivere dal punto di vista funzionale un server NFS è in genere molto semplice da configurare. Infatti di norma il servizio viene avviato automaticamente dagli script di avvio installati dai pacchetti, e tutto quello che c'è da configurare è il file `/etc/exports` che definisce su ciascuna macchina quali sono le directory da *esportare* verso l'esterno e quali sono le altre macchine che hanno la facoltà di utilizzarle; un esempio di questo file è il seguente:

/etc/exports
```
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
/home/piccardi/share   192.168.1.0/255.255.255.0(rw)
```

Il file prevede una serie di campi separati da spazi o tabulazioni; il primo campo specifica la directory sul server che verrà condivisa, i campi successivi definiscono chi e come può utilizzarla e si può specificare sia una singola stazione che una rete. Qualora si indichi una rete occorre specificarne il numero di bit (in notazione CIDR) o la maschera (in notazione *dotted decimal*) dopo il carattere "`/`". Bisogna però stare attenti perché non tutti i server supportano l'uso della notazione CIDR. Si possono usare anche indirizzi simbolici, che devono poter essere risolti, nel qual caso si possono usare i caratteri *wildcard* "*" e "?" per esprimere i nomi.

Per ciascuna riga di `exports` si possono indicare più macchine o reti separandoli con degli spazi, per ciascuna macchina e rete poi si potranno specificare delle modalità di accesso e delle

[26] il *file locking* è la capacità di bloccare ad altri processi l'accesso ad un file, per una descrizione dettagliata si veda il capitolo 11 di [GaPiL].

opzioni di funzionamento del protocollo indicandole fra parentesi tonde,[27] se si usano più opzioni queste devono essere separate da virgole; il formato generico di una riga del file `exports` sarà pertanto qualcosa del tipo:

```
/path/to/directory net1/mask1(option11,option12) machine2(option21,option22)
```

Le opzioni sono suddivise in due gruppi: quelle generali, relative al funzionamento del server e quelle di mappatura degli utenti. Le opzioni principali riguardanti il comportamento del server sono quelle che permettono di impostare le modalità di accesso ai file: `rw` e `ro`, che indicano rispettivamente lettura/scrittura e sola lettura, e quelle che ne governano la risposta in reazione a richieste di scrittura, come `sync` che richiede che la scrittura dei dati su disco sia completata prima di concludere una risposta, ed `async` che permette una risposta immediata con una scrittura dei dati asincrona. Questa opzione è disabilitata di default, in quanto in caso di crash del server le modifiche andrebbero perse; il comando `exportfs` notifica se nessuna di queste due opzioni è stata impostata, avvisando che sarà usata `sync` di default. Le principali opzioni ed il relativo significato sono riportati in tab. 8.13, l'elenco completo delle opzioni è riportato nella pagina di manuale accessibile con `man exports`.

Opzione	**Significato**
`rw`	accesso in lettura e scrittura.
`ro`	accesso in sola lettura.
`async`	abilita la scrittura asincrona sul server.
`sync`	esegue le scritture in maniera sincrona.
`root_squash`	rimappa l'UID ed il GID 0.
`no_root_squash`	non rimappa l'UID ed il GID 0.
`all_squash`	mappa tutti gli UID e i GID.
`anonuid`	imposta l'UID usato per l'accesso anonimo.
`anongid`	imposta il GID usato per l'accesso anonimo.

Tabella 8.13: Possibili opzioni per le directory esportate tramite NFS nel file `/etc/exports`.

Uno dei problemi principali che si deve affrontare quando si esporta una directory con NFS è quello della titolarità dei file. Infatti il protocollo associa i file agli utenti ed ai gruppi così come sono definiti sul server, utilizzando i relativi UID e GID. Un utente sul client potrà avere accesso ai *suoi* file condivisi solo se questi identificativi (questa volta come definiti sul client) corrispondono; è necessario perciò tenere sincronizzati UID e GID in modo che sia sui client che sul server essi corrispondano agli stessi utenti e gruppi. Per fare questo si può ricorrere all'uso di LDAP per la centralizzazione degli utenti (l'argomento è trattato in [IS-LDAP]) o utilizzare il servizio NIS.

In alcuni casi però occorrono delle eccezioni, ad esempio non è desiderabile che l'amministratore di un client sia trattato come *root* anche nell'accesso ai file del server. Per questo motivo di default è attiva l'opzione `root_squash` che rimappa gli UID e GID 0 usati dall'amministratore sul valore 65534 (-2), corrispondente rispettivamente a `nobody` e `nogroup`.[28]

[27] si faccia attenzione a non specificare le opzioni inserendo uno spazio fra il nome della macchina e le parentesi che le delimitano (è un errore comune); in tal caso infatti le opzioni verrebbero prese come l'indirizzo di una nuova macchina, con effetti fra l'imprevedibile ed il rischioso.

[28] quello del controllo sull'accesso ai file in base al solo valore dell'UID è un grosso problema di sicurezza di NFS; basta infatti collegarsi ad una rete con un IP consentito per far essere in grado, una volta che ci si sia dati

La presenza di una riga in /etc/exports non significa che la directory ivi indicata sia resa disponibile automaticamente; per controllare i filesystem che sono *esportati* è infatti necessario un apposito comando exportfs. Di norma lo si utilizza negli script di avvio per esportare tutti i filesystem elencati in /etc/exports con exportfs -a. Qualora si modifichi /etc/exports si può usare il comando exportfs -r per sincronizzare la tabella dei filesystem esportati con i contenuti del file. Infine con l'opzione -u si può rimuovere uno dei filesystem esportati, mentre con -o si possono cambiare le opzioni. I dettagli si trovano al solito nella pagina di manuale, accessibile con man exportfs.

Si tenga conto infine che i vari demoni che forniscono il servizio hanno il supporto per i TCP wrapper, quindi onorano le restrizioni descritte in sez. 8.1.5. È buona norma, vista la delicatezza del servizio fornito (che permette di accedere ai file degli utenti nelle directory condivise), impostare sempre una politica di accesso negato di default per tutti i demoni, per abilitare in /etc/hosts.allow gli accessi alle stesse macchine riportate in /etc/exports.

8.4.2 NFS sul lato client

Per poter utilizzare NFS sul lato client occorrerà avere il supporto nel kernel, ed oltre a portmap dovranno essere attivi i due demoni rpc.statd e rpc.lockd. Inoltre si deve avere una versione sufficientemente recente del comando mount che sia in grado di montare un filesystem NFS, ma al giorno d'oggi qualunque versione lo è.

Come per il lato server di norma l'utilizzo è molto semplice e si esaurisce nel montare il filesystem remoto sulla directory prescelta, in quanto con l'installazione dei relativi pacchetti tutte le distribuzioni si curano che siano avviati tutti i servizi necessari.

Montare un filesystem NFS è comunque estremamente semplice, invece di indicare un dispositivo basterà indicare l'indirizzo (simbolico o numerico) del server seguito da ":" e dalla directory che si vuole montare, che dovrà essere presente e accessibile nel file /etc/exports del server. Se si vuole eseguire il comando a mano basterà eseguire mount con una sintassi del tipo:

```
mount -t nfs indirizzoserver:/directory/esportata /mount/point
```

e si può poi definire un *mount point* permanente in /etc/fstab aggiungendo una riga del tipo:

```
indirizzoserver:/directory/esportata /mount/point nfs user,exec,noauto 0 0
```

dove la differenza con un filesystem su disco è l'uso di nfs come *filesystem type* e l'indirizzo della directory da montare al posto del file di dispositivo.

Nell'esempio si sono usate soltanto delle opzioni di montaggio generiche ma in realtà NFS supporta una serie di opzioni specifiche che ne controllano il funzionamento. Buona parte delle opzioni di NFS devono essere specificate nella forma opzione=valore, fanno eccezione le opzioni che prendono valori logici, che si specificano direttamente per attivarle, o con il prefisso no per disattivarle.

Due opzioni, hard e soft controllano il comportamento del client in assenza di connessione al server o di crollo di quest'ultimo; in questo caso infatti si pone il problema di cosa fare per i processi che stanno usando o cerchino di usare i file condivisi su una directory che è stata montata via NFS.

---

l'UID corrispondente, di leggere i file di un qualunque utente; per questo motivo dalla versione 4 del protocollo è possibile appoggiarsi ad un server Kerberos per autenticare in maniera forte sia i client che gli utenti.

Quello che succede normalmente è che in questo caso il processo viene posto in stato `D` (*uninterruptible sleep*, vedi tab. 1.10) ed il client continua ad interrogare il server fino a quando questo non ritorna disponibile e l'esecuzione riprende (dal punto di vista del processo) come se niente fosse accaduto. Si tenga presente che essendo in stato `D` non sarà possibile terminare i processi rimasti bloccati; se si vuole poter uccidere i processi con `kill` occorrerà usare l'opzione `intr`.

Con una directory montata con il valore di default, `hard`, il client prosegue nel tentativo di riconnessione al server senza limiti di tempo ed i processi restano bloccati indefinitamente, se invece si è usato `soft` dopo un certo periodo di tempo di tentativi di riconnessione falliti, verrà notificato al processo che ha eseguito l'accesso un errore di I/O su disco. Questo intervallo di tempo, che viene detto un *major timeout*, dipende dalle altre impostazioni di NFS, ed in particolare dalle opzioni di montaggio `timeo` e `retrans`.

Opzione	**Significato**
`rsize`	imposta le dimensioni del buffer di lettura, da indicare in numero di byte, il default è 4096.
`wsize`	identica a `rsize` ma per il buffer di scrittura.
`hard`	in caso di crollo del server continua indefinitamente i tentativi di riconnessione.
`soft`	in caso di crollo del server i tentativi di riconnessione vengono effettuati per un periodo limitato di tempo e falliscono definitivamente dopo il numero di tentativi fissato con `retrans`.
`intr`	permette di inviare segnali ai processi bloccati per un crollo del server, con i kernel più recenti (posteriori al 2.6.35) questa opzione viene ignorata ed occorre usare semplicemente il segnale `SIGKILL`.
`timeo`	il tempo (in decimi di secondo) atteso per una risposta prima di effettuare la ritrasmissione di una richiesta, è di 60 secondi per connessioni TCP e variabile a seconda delle operazioni per UDP, viene aumentato ogni volta che una ritrasmissione (raddoppiandolo) scade di nuovo fino ad un massimo di 600 secondi per TCP e 60 secondi per UDP.
`retrans`	il numero di tentativi di ritrasmissione che vengono effettuati prima di generare un avviso, il default è tre volte, e se si è usato `hard` il ciclo verrà ripetuto indefinitamente, se si è usato `soft` il client darà un errore definitivo.
`nolock`	disabilita il *file locking* su NFS; viene usato per compatibilità con alcuni vecchi server che non lo supportano e non è utilizzabile con NFS4.
`tcp`	usa il protocollo TCP.
`udp`	usa il protocollo UDP (il default).
`sec`	specifica le modalità di autenticazione presso il server, con `sys` (il default) vengono usati i valori locali di UID e GID, con `krb5` viene usato Kerberos per l'autenticazione.

Tabella 8.14: Opzioni del comando `mount` specifiche per NFS.

Altre due opzioni importanti per le prestazioni sono `rsize` e `wsize` che indicano le dimensioni (in byte) dei buffer di lettura e scrittura. Il valore di default è di 4096 per entrambe, ma ovviamente più ampio è il valore migliori saranno le prestazioni. Le altre opzioni principali sono illustrate in tab. 8.14. Per ulteriori dettagli e l'elenco completo delle opzioni si può consultare la apposita pagina di manuale, accessibile con `man nfs`.

Si ricordi infine, come accennato in sez. 8.4.1, che i proprietari dei file sulle directory montate via NFS sono riconosciuti in base ai valori di UID e GID definiti sul server. Pertanto se sul client

l'utente relativo non esiste o ha un UID diverso (cosa possibile, se non si sono creati gli utenti nella stessa sequenza) la titolarità dei file non corrisponderà, e si avranno problemi.

Con la versione 4 di NFS è possibile appoggiarsi ad un server Kerberos,[29] sia per quanto riguarda l'identificazione e l'autenticazione di utenti e macchine, che per la verifica e anche la cifratura del traffico NFS. Si tenga presente infatti che di default il traffico dati di NFS non è né cifrato né autenticato, ed è quindi poco sicuro; con NFS4 e Kerberos questi problemi possono essere risolti, al prezzo di una maggiore complessità di configurazione e prestazioni ovviamente inferiori.

8.4.3 La configurazione di base di Samba

La suite Samba è un insieme di programmi che permette ad una macchina Unix di utilizzare il protocollo CIFS (*Common Internet File System*) usato dai server Windows per la condivisione dei file. In realtà Samba permette di fare molto di più, e può ad esempio sostituire un PDC (*Primary Domain Controller*) Windows, ma qui noi ne tratteremo solo le caratteristiche di base relative alla condivisione dei file.

Il nome Samba deriva dalla precedente denominazione di CIFS che veniva chiamato in maniera non ufficiale SMB (*Service Message Block*), e l'autore racconta di aver scelto il nome eseguendo il comando `grep s.m.b.` sul file del dizionario. Nelle sue prime versioni il servizio veniva fornito attraverso il protocollo proprietario *NetBIOS* usato da Windows come protocollo di trasporto; a questo livello il protocollo ormai è in disuso, ed attualmente è disponibile direttamente su TCP/IP.

L'implementazione dei servizi SMB è realizzata da Samba attraverso due demoni, `smbd` che implementa la condivisione dei file e delle stampanti e `nmbd` che implementa i servizi NetBIOS (cioè il riconoscimento della presenza di server sulla rete, la risoluzione dei nomi, ecc.). L'avvio del servizio è generalmente curato dagli opportuni script; nel caso di Debian ad esempio questa viene gestito da `/etc/init.d/samba` per entrambi i demoni, altre distribuzioni usano due script separati.

Entrambi i demoni vengono controllati tramite un unico file di configurazione, `smb.conf`. La sintassi è sempre analoga a quella dei file `.ini` di Windows che abbiamo illustrato in sez. 3.1.1. Un estratto di questo file, preso dalla installazione di default di una Scientific Linux, è il seguente:

/etc/samba/smb.conf

```
[global]
        workgroup = MYGROUP
        server string = Samba Server Version %v
        log file = /var/log/samba/log.%m
        max log size = 50
        security = user
        passdb backend = tdbsam
        load printers = yes
        cups options = raw
[homes]
        comment = Home Directories
        browseable = no
        writable = yes
[printers]
```

[29] la trattazione di Kerberos va al di là dello scopo di questo testo, un buon libro sull'argomento è [Kerberos].

```
comment = All Printers
path = /var/spool/samba
browseable = no
guest ok = no
writable = no
printable = yes
```

Il file è diviso in sezioni distinte introdotte da un nome in parentesi quadra, ciascuna delle quali descrive una risorsa condivisa attraverso SMB (quello che viene chiamato uno *share*). Sono però previste tre sezioni speciali, `[global]`, `[homes]`, e `[printers]` che non corrispondono a degli *share*, ma servono ad impostare alcune funzionalità generiche. All'interno di una sezione i vari parametri sono impostati con direttive del tipo `parola chiave = valore`.

La sezione `[global]` permette di impostare i parametri che si applicano al server nel suo insieme. Le direttive fondamentali sono `workgroup`, che definisce il nome del dominio NT in cui figurerà la macchina, e `security`, che definisce le modalità di accesso al server e può assumere i valori illustrati in tab. 8.15.

Valore	**Significato**
`user`	gli utenti devono autenticarsi presso il server, ed a ciascuno di essi deve corrispondere un utente Unix, questo comporta che gli utenti anonimi non funzioneranno a meno di non impostare opportunamente la direttiva `map to guest`.
`domain`	gli utenti vengono autenticati presso un *server di dominio* esterno, questo però richiede che si sia effettuata una unione della propria macchina allo stesso con l'appropriato uso del comando `net`.
`ads`	gli utenti vengono autenticati presso un server *Active Directory*, la cosa richiede installazione e configurazione locale di Kerberos oltre all'unione della macchina al dominio *Active Directory* con l'appropriato uso del comando `net`.
`share`	opzione deprecata, utilizzabile in pratica solo quando si vuole solo concedere un accesso anonimo (consente di evitare l'invio della password da parte del client), non funziona con le versioni recenti del protocollo (SMB2).
`server`	l'autenticazione viene dirottata verso un altro server SMB (senza dominio) e se questa fallisce ricade nel caso di `user`.

Tabella 8.15: Possibili valori della direttiva `security` in `/etc/samba/smb.conf`.

Di norma, a meno di non avere soltanto un accesso anonimo, nel qual caso si può usare `share`, si utilizza come valore `user` che richiede che esista un utente Unix per ciascun utente che si collega al server. I valori `domain`, `server` e `ads` servono quando si vuole dirottare la richiesta di autenticazione (caso che non tratteremo in questa sede). In ogni caso Samba effettua gli accessi ai file per conto di un utente UNIX locale, con i permessi associati a quest'ultimo, anche nel caso di accesso anonimo, nel qual caso viene usato l'utente indicato dalla direttiva `guest user`.

In tab. 8.16 sono riportati le altre direttive principali. L'elenco completo di tutti i parametri è descritto in dettaglio nella sezione `PARAMETERS` della pagina di manuale di `smb.conf`, di norma nel file fornito dai pacchetti di installazione, vengono impostati valori ragionevoli adatti agli usi più comuni, la trattazione dettagliata degli stessi va al di là dello scopo di questo testo, ma per maggiori informazioni si può consultare il *Samba-HOWTO*, inoltre tutti i dettagli delle direttive di configurazione sono documentati nella pagina di manuale di `smb.conf`.

Direttiva	**Significato**
`dns proxy`	quando il server è usato per la risoluzione dei nomi, passa le richieste non risolte al DNS.
`encrypt passwords`	utilizza una autenticazione basata su password cifrate, l'impostazione di default è `true` in quanto corrispondente al rispettivo default di Windows 98 e NT; se attivato necessita la presenza dei dati di autenticazione in un opportuno supporto, in genere tramite `smbpasswd`.
`guest account`	specifica l'utente utilizzato per l'accesso *ospite* (cioè senza autenticazione); di norma si usa `nobody` che non ha nessun privilegio (in certi casi questo non consente l'uso delle stampanti).
`map to guest`	viene usato per indicare al servizio cosa fare quando viene fatta una richiesta per un utente che non è presente nel database (e quindi non ha un utente Unix corrispondente) o non può essere autenticato; prevede quattro valori: `never` in cui gli accessi vengono sempre rigettati (il default); `bad user` in cui viene rigettato l'accesso in caso di password sbagliata mentre gli utenti inesistenti vengono mappati sul `guest account`; `bad password` in cui anche gli accessi con password sbagliata vengono mappati sul `guest account`, `bad uid`, possibile solo con `security` impostato a `ads` o `domain` in cui gli accessi con utenti non riconosciuti nel dominio vengono mappati sul `guest account`.
`passdb backend`	specifica la lista dei supporti dove vengono mantenute le password, il default è `tdbsam`, che indica un database binario, per usare un file testuale si usa `smbpasswd` mentre `ldapsam` ricava i dati da un server LDAP.
`printing`	indica come devono essere ricavate le informazioni riguardo le stampanti condivise da Samba, prevede diversi valori per i vari server di stampa, nel caso di Linux viene usato di default il valore `cups` che indica di appoggiarsi ad un server CUPS (vedi sez. 3.4.2).
`security`	stabilisce le modalità di accesso al server, il valore di default è `user` che richiede un username ed una password per garantire l'accesso; è necessaria la presenza di un corrispondente utente sulla macchina.
`socket options`	specifica delle opzioni relative al comportamento del sistema nei confronti della rete.
`unix password sync`	se specificato il server tenta di sincronizzare le password UNIX quando si è cambiata quella su Windows.
`wins support`	indica se supportare la risoluzione dei nomi *NetBIOS* di NT.
`wins server`	indica un server per la risoluzione dei nomi *NetBIOS* usati dal protocollo.
`workgroup`	definisce il gruppo di lavoro di cui il server farà parte (quello che è il nome di dominio NT, detto anche *workgroup name*).

Tabella 8.16: Significato delle principali direttive globali di Samba.

La sezione `[homes]` quando dichiarata permette di creare al volo degli *share* corrispondenti alle home degli utenti. In caso di connessione prima vengono controllate le altre sezioni definite, e qualora non sia stata trovata nessuna corrispondenza la sezione richiesta viene trattata se fatta verso una condivisione di rete il cui nome corrisponde allo username dell'utente. Se l'utente esiste ed è stato correttamente autenticato (torneremo su questo a breve) viene reso accessibile al volo il corrispondente *share* con le caratteristiche impostate in questa sezione.

La sezione `[printers]` ha lo stesso scopo della sezione `[homes]`, ma vale per le stampanti, se non viene trovata una sezione esplicita corrispondente alla stampante cercata, vengono esaminate le informazioni relative alle stampanti (in modalità diverse a seconda di quanto indicato con la direttiva `printing`) per verificare se esiste una stampante con il nome richiesto, ed in caso positivo viene creato al volo il relativo *share*.

Per aggiungere uno *share* è sufficiente definire una nuova sezione e per creare un *share* basterà

inserire in coda al file mostrato in precedenza una ulteriore sezione del tipo:

```
/etc/samba/smb.conf
[foo]
path = /home/bar
browseable = yes
read only = no
```

in questo modo gli utenti potranno accedere ad uno *share* denominato `foo` corrispondente al contenuto della directory `/home/bar`.

La direttiva principale in questo caso è `path`, usabile solo nella dichiarazione di uno *share*, che consente di indicare quale directory della propria macchina verrà resa accessibile attraverso di esso. Con la configurazione indicata l'accesso richiederà sempre un username ed una password, a meno che non si indichi in una ulteriore riga la direttiva `guest ok = yes`, nel qual caso sarà consentito l'accesso anonimo senza password, che avverrà per conto dell'utente specificato da `guest user`.

In ogni caso gli effettivi permessi disponibili sulla directory dello *share* (`/home/bar` nel caso dell'esempio) saranno quelli previsti dal sistema per l'utente con il quale ci si è connessi al server; Samba non può in nessun caso garantire maggiori permessi di quelli forniti dal sistema, può invece ridurli, ad esempio impostando `read only = yes` per consentire un accesso in sola lettura.

L'uso della direttiva `browseable = yes` permette di far apparire lo *share* nella lista pubblica mostrata dal cosiddetto *Network Neighborhood* di Windows, cioè di farlo visualizzare quando si richiede di "*sfogliare la rete*". Le altre principali direttive utilizzabili nella definizione di uno *share* sono riportate in tab. 8.17, per i dettagli e l'elenco completo si rimanda di nuovo alla documentazione ufficiale di Samba. Il pacchetto Samba mette comunque a disposizione il comando `testparm`, che esegue una analisi sommaria del file di configurazione in grado di rilevare gli errori più macroscopici.

In generale per l'accesso ai servizi di condivisione forniti da Samba, che siano definiti esplicitamente come appena visto, o implicitamente con la sezione `[homes]`, è sempre necessaria l'autenticazione. Il problema è che Windows usa un meccanismo di autenticazione e gestione degli utenti completamente diverso e non compatibile con quello nativo di Linux.

Il primo punto è che la procedura di autenticazione, dovendo passare sulla rete, per avere un minimo di sicurezza non può prevedere il passaggio diretto di credenziali che potrebbero essere intercettate, ed inoltre i client Windows non sono in grado di inviare password in formato UNIX. Per questo non è possibile eseguire un confronto diretto con il contenuto di `/etc/passwd` come avviene per l'autenticazione locale.[30]

Questo comporta che Samba deve essere in grado di effettuare autonomamente l'autenticazione utilizzando lo stesso protocollo usato da Windows, da cui segue automaticamente la necessità di mantenere un elenco di utenti e password, queste ultime cifrate in maniera adeguata ai protocolli usati da Windows, in maniera indipendente da quelle usate per gli utenti ordinari del sistema ospite.

[30] in realtà la cosa è possibile usando la procedura di autenticazione obsoleta in cui la password viene inviata in chiaro ed il server potrebbe operare il confronto; questo è quello che avviene quando si forza il server a lavorare con le password trasmesse in chiaro ponendo il parametro `encrypt password` a `false`, ma questa modalità consente un uso limitato ed è comunque da evitare per gli ovvi problemi di sicurezza.

Direttiva	**Significato**
`browseable`	indica se lo *share* deve essere pubblicato sulla rete, prende i valori `yes` o `no` (il default è `no`).
`create mask`	maschera dei permessi per la creazione dei file, i bit non indicati verranno rimossi, in genere si indica con un valore ottale ed il suo default è `0644`.
`directory mask`	analoga di `create mask` per le directory.
`host allow`	indica la lista delle stazioni da cui è possibile connettersi al server: si possono usare sia indirizzi IP che nomi a dominio, con notazione sia CIDR che *dotted decimal*; la direttiva si può utilizzare anche globalmente e si applicherà a tutti gli *share*.
`guest ok`	indica se è consentito l'accesso anonimo, prende i valori `yes` o `no` (il default è `no`).
`host deny`	analogo del precedente, indica la lista delle stazioni da cui non è possibile connettersi al server, in caso di conflitto con `host allow` quest'ultimo ha la precedenza.
`invalid users`	indica la lista degli utenti per conto dei quali non si può utilizzare il servizio: di norma si specifica sempre *root* o gli altri utenti di sistema e la si usa solo a livello globale ed ha la precedenza su `valid users`.
`read only`	indica se l'accesso è in sola lettura, prende i valori `yes` o `no` (il default è `yes`).
`path`	indica il pathname della directory in cui sono contenuti i file dello *share* (o la coda di stampa per una stampante), si possono utilizzare espressioni di sostituzione come `%u` per il nome dell'utente o `%m` per il nome della macchina e `%S` per il nome dello *share*.
`valid users`	indica la lista degli utenti che possono accedere al servizio, per tutti gli altri sarà negato; si possono indicare più nomi separati da spazi ed un nome che inizia con "`@`" verrà interpretato come nome di un gruppo e l'accesso sarà garantito a tutti i suoi membri, il default è vuoto che indica l'accesso a tutti gli utenti.

Tabella 8.17: Le principali direttive di Samba per gli *share*.

Il secondo punto è quello relativo al fatto che nel mondo Windows agli utenti viene associato un identificativo unico, il cosiddetto *SID*, che può essere usato anche per gruppi e per macchine all'interno di quello che viene chiamato un *dominio*. Su Unix utenti e gruppi sono separati e non esiste il concetto di un account associato ad una macchina.[31] Dato che la scelta fondamentale dell'architettura di Samba è che gli accessi ai file resi accessibili su uno *share* vengano sempre fatti per conto di un utente della macchina, questo comporta che per gestire gli utenti Windows deve essere creata una opportuna corrispondenza con utenti Unix.

In realtà le modalità con cui vengono gestiti gli utenti dipendono dalla modalità in cui si sta utilizzando Samba, dato che, come accennato, è sempre possibile delegare la procedura di autenticazione ad un'altra macchina. Qui però prenderemo in considerazione solo il caso in cui gli utenti e l'autenticazione vengono gestiti direttamente sul proprio server, supporremo cioè che si stia usando Samba avendo impostato `security = user`.

I supporti usati da Samba per mantenere i dati di autenticazione e le corrispondenze citati sono vari, il più semplice, rimasto a lungo il default, è quello di registrarli su un file di testo analogo di `/etc/passwd`. Questo viene abilitato specificando per il parametro di configurazione `passdb backend` il valore `smbpasswd`, ed il file da usare può essere specificato di seguito con

[31] questo significa ad esempio che quando si usa Samba come *Domain Controller*, cosa che non tratteremo in questa sede, per ciascuna macchina nel dominio deve essere disponibile un opportuno utente locale.

il relativo pathname, anche se di solito non si specifica niente e viene usato il default che è `/var/lib/samba/smbpasswd`.

Questo file contiene gli *hash* in formato Windows delle password di ciascun utente. Il formato del file è analogo a quello di `passwd`, una riga per ogni utente con campi separati dal carattere "`:`", il primo campo specifica il nome utente ed il secondo il suo UID che devono corrispondere agli analoghi in `passwd`. Seguono due *hash* crittografici nel formato usato da Windows, poi dei flag per l'*account*, identificati da lettere fra parentesi quadra e infine la data di ultima modifica della password. Il formato del file è descritto in dettaglio nella pagina di manuale accessibile con `man 5 smbpasswd`.

Un secondo supporto, usato di default nelle versioni recenti di Samba, è quello di *tinydb* un piccolo database binario, che permette di velocizzare notevolmente le operazioni di scansione. In tal caso si dovrà indicare per `passdb backend` il valore `tdbsam` e i dati verranno memorizzati nel file `/var/lib/samba/passdb.tdb`, a meno di non aver specificato un pathname diverso come parametro opzionale di `tdbsam`. È infine possibile usare come supporto un server LDAP, nel qual caso il valore da usare è `ldapsam` seguito dalla URL che indica il server da contattare.[32]

In generale Samba supporta la gestione degli utenti attraverso il protocollo CIFS utilizzando il comando `net` che costituisce l'analogo dell'omonimo programma presente su Windows. Questa modalità è completamente generica e consente di gestire gli utenti anche se questi sono gestiti su un server remoto. Noi però non tratteremo questo comando, ma quelli che operano localmente agendo direttamente sui supporti citati.

Di norma la creazione dei dati per l'autenticazione degli utenti Windows deve essere eseguita esplicitamente dall'amministratore, ed i singoli utenti devono essere esplicitamente definiti per Samba. Non essendo infatti possibile ricavare la password in chiaro degli utenti dagli *hash* crittografici mantenuti in `/etc/passwd` (o da qualunque altro supporto usato dal *Name Service Switch*) non esiste una modalità di "*replicazione*" automatica dei contenuti di quest'ultimo per generare automaticamente gli utenti Windows da quelli Unix creando gli *hash* delle password nel formato usato da Samba. Si ricordi che comunque per poter creare un utente Windows deve essere già definito nel sistema l'utente Unix su cui questo verrà rimappato.

Per questo insieme al server viene fornito anche il comando `smbpasswd` che permette l'inserimento dei dati di autenticazione all'interno del sistema utilizzato da Samba; il comando userà quanto specificato con la direttiva `passdb backend`, e l'amministratore dovrà aggiungere i singoli utenti a mano con qualcosa del tipo:

```
smbpasswd -a utente
```

in questo caso verrà chiesta la password (due volte per evitare errori di battitura) e l'utente, che in questo caso deve corrispondere ad uno già presente nel sistema come utente Unix, verrà aggiunto all'interno dell'opportuno supporto usato da Samba.

Un utente normale può usare lo stesso comando sul suo username per modificare la sua password, nel qual caso dovrà comunque fornire quella precedente. Solo l'amministratore può operare per conto di altri utenti; l'opzione `-a` vista nell'esempio è quella che permette di aggiungere un utente, mentre l'opzione `-x` permette di cancellarlo. Con le opzioni `-e` e `-d` è invece possibile abilitare e disabilitare temporaneamente un utente, mentre con `-n` si può impostare una password vuota. Al solito l'elenco completo delle opzioni è disponibile nella pagina di manuale del comando, accessibile con `man smbpasswd`.

[32] questa configurazione è trattata con dovizia di particolari in sez. 2.2.4 di [IS-LDAP].

Oltre al server per fornire a client Windows i servizi di condivisione di una macchina Unix, la suite di applicativi realizzata dal progetto Samba comprende anche alcuni programmi che permettono di utilizzare lato client da un sistema di tipo Unix i servizi presenti su un server Windows o su un altro server Samba, anche se in quest'ultimo caso sarebbe più opportuno utilizzare le applicazioni native Unix.

Il primo di questi programmi è `smbclient`, che permette di connettersi ad uno *share* di un server Windows con una interfaccia simile a quella di FTP. Il comando prende come argomento il nome di un servizio Windows a cui collegarsi nella forma generica:

```
//nome_server/nome_servizio
```

dove `nome_server` è il nome della macchina secondo il protocollo CIFS (quello che viene chiamato *NetBIOS name*) e `nome_servizio` il nome della condivisione (o della stampante) su quella macchina. Si noti come il nome sia analogo a quello che viene usato da Windows, usando però il carattere di separazione "`/`" al posto di "`\`", ma è possibile usare anche quest'ultimo, solo che si deve tener conto che la shell lo interpreta, per cui nella riga di comando deve essere specificato due volte, rendendo le cose meno leggibili.

L'opzione principale è `-U` che permette di specificare il nome utente (relativo al server) con il quale ci si vuole connettere al servizio, se non lo si specifica viene usato il proprio username. La password può essere scritta direttamente sulla riga di comando come secondo argomento (o specificando l'username nella forma `utente%password`, altrimenti verrà chiesta dal comando stesso sul terminale senza mostrarla. Se il servizio a cui si accede non richiede una password se ne può eliminare la richiesta con l'opzione `-N`, senza `-N` il comando chiede comunque una password, anche quando questa non è richiesta.

Una forma alternativa del comando è quella in cui si usa soltanto l'opzione `-L` per richiedere una lista dei servizi visibili presenti su un server, il cui nome deve essere passato come parametro dell'opzione. Conviene in genere abbinarla a `-N` (per l'operazione non è richiesta una password), così da ottenere un risultato del tipo:

```
piccardi@hain:~$ smbclient -N -L 192.168.1.129
Domain=[TEST] OS=[Unix] Server=[Samba 3.5.6]

        Sharename       Type      Comment
        ---------       ----      -------
        IPC$            IPC       IPC Service (squeeze server (Samba 3.5.6))
        PROVA           Disk
        PRINT$          Disk      Printer Drivers
        lpcolor         Printer   Color Laserjet 2605dn
        lp              Printer   HP LaserJet 1320N
Domain=[TEST] OS=[Unix] Server=[Samba 3.5.6]

        Server               Comment
        ---------            -------
        SQUEEZE              squeeze server (Samba 3.5.6)

        Workgroup            Master
        ---------            -------
        TEST                 SQUEEZE
```

Se invece si usa `smbclient` per accedere ad un servizio una volta completata l'autenticazione si verrà portati su una riga di comando che supporta una sintassi simile a quella di FTP, con

gli stessi comandi di tab. 7.37. Per gli altri i dettagli relativi all'uso del comando e per l'elenco completo delle altre opzioni si può fare riferimento alla rispettiva pagina di manuale, accessibile con `man smbclient`, che riporta anche vari esempi di utilizzo.

Una seconda modalità per accedere ai dati di uno *share* è quella di montarne il contenuto all'interno del filesystem esattamente come si fa per NFS. Questo richiede il supporto nel kernel, di solito presente in tutte le distribuzioni standard, e comunque attivabile nella sezione `Network filesystem` dei menu di configurazione. In realtà esistono due diversi filesystem di rete che supportano il protocollo CIFS, il vecchio `smbfs`, ormai deprecato e quasi universalmente sostituito dal nuovo `cifs`.[33]

Tradizionalmente per montare uno *share* era previsto l'uso esplicito del comando `smbmount` la cui sintassi è:

```
smbmount service mount-point [ -o options ]
```

dove `service` identifica lo *share* da montare nella stessa forma usata per `smbclient`. Il comando è sostanzialmente equivalente all'uso di `mount` con l'opzione `-t smbfs`. Era anche previsto un analogo comando `smbumount` per smontare un filesystem `smbfs`.

Ormai questo comando è deprecato e nell'uso attuale viene utilizzato solo il comando `mount.cifs`, equivalente, come visto in sez. 5.1.3, all'uso di `mount -t cifs`. Il comando segue la sintassi ordinaria di `mount` ma invece che un file di dispositivo prevede che si passi come primo argomento l'identificato dello *share*, anche in questo caso con la stessa sintassi usata per `smbclient`.

Il comportamento del comando viene controllato dalle opzioni di montaggio, che si passano al solito come parametri dell'opzione `-o` del comando come lista di assegnazioni nella forma `opzione=valore` separate da virgole. Le opzioni più importanti sono quelle che consentono di passare le credenziali di autenticazione; queste possono essere specificate direttamente sulla riga di comando con le opzioni `user` e `password`, anche se l'opzione `user` consente l'indicazione contemporanea di entrambe con la stessa sintassi dell'opzione `-U` di `smbclient`.

Dato che si ha a che fare con un accesso generico ad uno *share* Windows l'indicazione di un utente ed una password può non essere sufficiente dato che in alcuni casi su una rete possono essere presenti più *domini* per cui l'indicazione dell'utente deve anche indicare a quale di questi domini si fa riferimento, nella forma `dominio/utente`. Inoltre siccome la riga di comando può essere mostrata (con la password che vi si è scritta) da `ps` è in genere preferibile inserire tutte queste credenziali in un file (opportunamente protetto da letture indesiderate) ed usare l'opzione `credentials` per indicarne il *pathname*.

Inoltre come per NFS si pone il problema di come gestire la proprietà dei file e delle directory presenti nello *share*. Se esiste un meccanismo che consente di mappare gli utenti Windows sugli utenti locali della macchina, questa viene gestita automaticamente, ma non è detto questo sia sempre possibile, per cui in tal caso si possono usare le opzioni `uid` e `gid` per forzare una assegnazione statica. Si sono riportate altre principali opzioni di montaggio in tab. 8.18, per l'elenco completo ed i dettagli si consulti al solito la pagina di manuale, accessibile con `man mount.cifs`.

Infine per il monitoraggio delle connessioni al server è disponibile il comando `smbstatus` inoltre permette di controllare lo stato delle connessioni, degli utenti presenti e dei file utilizzati sul proprio server.

[33]si sono identificati le due diverse implementazioni utilizzando i nomi che devono essere indicati come parametro per l'opzione `-t` di `mount`.

Opzione	Significato
`user`	utente da usare per l'autenticazione, supporta l'indicazione anche della password e del dominio usando la forma generica `dominio/utente%password`.
`password`	password da usare per l'autenticazione.
`credentials`	file da usare per leggere le credenziali di autenticazione in modo che non compaiano sulla riga di comando, prevede la presenza della assegnazione di altrettanti valori alle opzioni `user`, `password` e `domain` (quest'ultima opzionale) su altrettante righe distinte.
`uid`	utente a cui assegnare la proprietà di file e directory.
`gid`	gruppo a cui assegnare la proprietà di file e directory.
`domain`	dominio a cui fare riferimento.
`guest`	richiede un accesso anonimo senza autenticazione.

Tabella 8.18: Le principali opzione di montaggio per il filesystem *cifs*.

8.5 La gestione elementare della posta elettronica

Tratteremo in questa ultima sezione le problematiche di base della gestione della posta elettronica, e dopo una introduzione generica al funzionamento del servizio tratteremo quanto riguarda le impostazioni da effettuare su una macchina per la ricezione ed il reinoltro della posta locale ed i comandi di base per la verifica del funzionamento e della spedizione della stessa.

8.5.1 La struttura del servizio della posta elettronica

Come per la posta ordinaria, quella che in gergo informatico viene detta *snail mail*, anche il servizio di gestione della posta elettronica prevede la presenza di diverse infrastrutture deputate alla spedizione, all'inoltro, alla ricezione e allo smistamento dei vari messaggi.

Il meccanismo è analogo a quello della posta ordinaria, un utente che invia la posta (il mittente, o "*sender*") deve utilizzare un apposito programma, denominato *Mail User Agent* (o MUA), che serve ad "*imbucare*" la propria posta elettronica presso un altro programma, denominato *Mail Transport Agent* (o MTA), che è quello che si incarica del trasporto della stessa fino al destinatario (il cosiddetto *recipient*) svolgendo cioè il ruolo del servizio postale. Per questo in genere si parla di "*client di posta*" per indicare un *Mail User Agent* e di "*server di posta*" per indicare un *Mail Transport Agent*, anche se vederemo che in realtà per gestire la posta elettronica oltre agli MTA esistono altri tipi di server, dedicati ad altri compiti.

Il caso più semplice di gestione della posta elettronica, illustrato in fig. 8.1, è il cosiddetto *trasporto locale* (o *local delivery*) in cui all'interno di una macchina si vuole inviare un messaggio ad un altro utente della macchina stessa. Questo caso è tipico del funzionamento ordinario dei sistemi Unix in cui alcuni servizi, come *cron*, usano proprio la posta elettronica per eseguire delle notifiche agli utenti.

Un qualunque MTA deve essere in grado di eseguire questo compito, ed in effetti nel corso del tempo sono stati sviluppati diversi server a questo scopo. Il primo di questi, ancora molto usato è *Sendmail*, che però, essendo nato come progetto cresciuto disordinatamente a cui sono state aggiunte progressivamente estensioni su estensioni ha sofferto a lungo di scarsa gestibilità del codice con gravi problemi di sicurezza ed è tuttora afflitto da una complessità eccessiva nella configurazione.

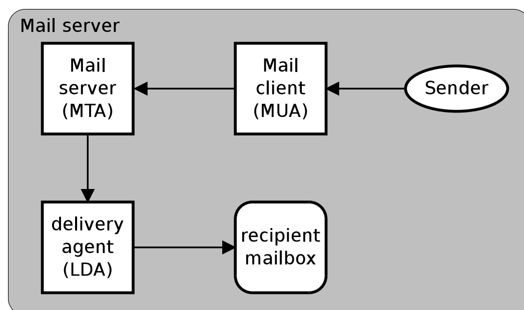


Figura 8.1: La posta elettronica nel caso di *trasporto locale*.

Per questo a vario titolo sono stati sviluppati altri server MTA, come *Exim*, *Qmail*, *Courier MTA*, *Postfix*, che permettessero di superare i vari limiti di *Sendmail*. Inoltre nel caso si voglia effettuare soltanto un inoltro locale esistono degli MTA, come `ssmtp`, dedicati allo svolgimento esclusivo di questo compito, che si possono installare tutte le volte che non sono necessarie le altre funzionalità e si deve ridurre al minimo l'uso delle risorse sulla macchina.

Inoltre per l'esecuzione dell'inoltro locale può entrare in gioco un ulteriore componente, anch'esso illustrato in fig. 8.1, detto *Local Delivery Agent* o LDA, che si incarica di svolgere il ruolo del postino nella consegna finale del messaggio nella *casella postale* del destinatario, e che può essere utilizzato per compiere delle ulteriori funzioni nella consegna finale del messaggio, come ad esempio suddividere i messaggi in cartelle a seconda dell'argomento, del mittente, scartare messaggi indesiderati, ecc.

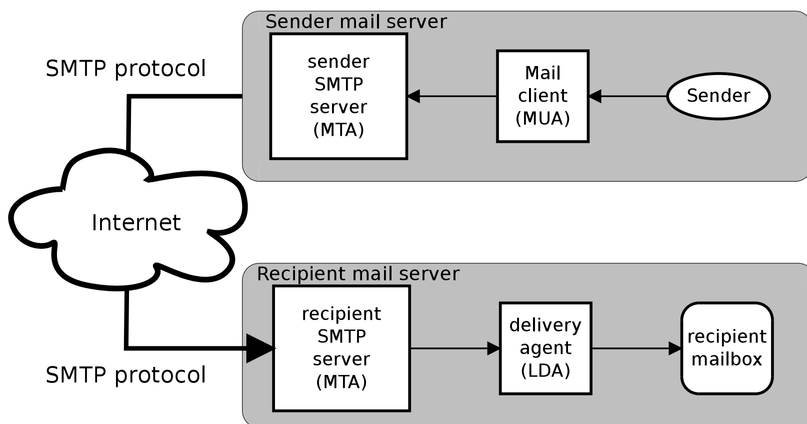


Figura 8.2: La posta elettronica nel caso di invio diretto da un server di posta.

In generale però il compito principale di un *Mail Transport Agent* è quello di curare il trasporto della posta per i destinatari remoti, ed è questa la parte più complessa. In genere questo compito viene svolto attraverso la rete come illustrato in fig. 8.2, contattando, con gli opportuni protocolli, il server di posta del destinatario, a cui viene poi delegata la consegna della

posta alla casella postale di quest'ultimo.

Il protocollo più utilizzato per il trasporto della posta è il cosiddetto *Simple Mail Transfer Protocol* (SMTP) attraverso il quale un server MTA può inviare i messaggi da consegnare ad un altro server MTA, o ricevere gli stessi da un client. Esistono comunque delle alternative, ad esempio prima della diffusione di internet, quando la trasmissione dei dati veniva effettuata attraverso collegamenti temporanei via modem, la posta veniva trasferita appoggiandosi ad altri protocolli come UUCP.[34]

Nel caso illustrato in fig. 8.2 il *Mail Transport Agent* ed il *Mail User Agent* sono semplicemente due programmi residenti sulla stessa macchina e possono quindi anche parlarsi direttamente. Vedremo in sez. 8.5.2 come sia possibile inviare messaggi direttamente con comandi generici indipendenti dal server MTA che si sta usando, che a seconda dei casi possono anche immetterli direttamente sulle code di spedizione di quest'ultimo.

Questa configurazione, che era quella classica utilizzata agli albori di internet, quando la posta si poteva spedire soltanto avendo un account su un server collegato alla rete, oggi viene usata sempre meno, dato che la posta si spedisce direttamente lavorando sulla propria stazione di lavoro, sulla quale normalmente non viene installato un server per la ricezione diretta della posta.

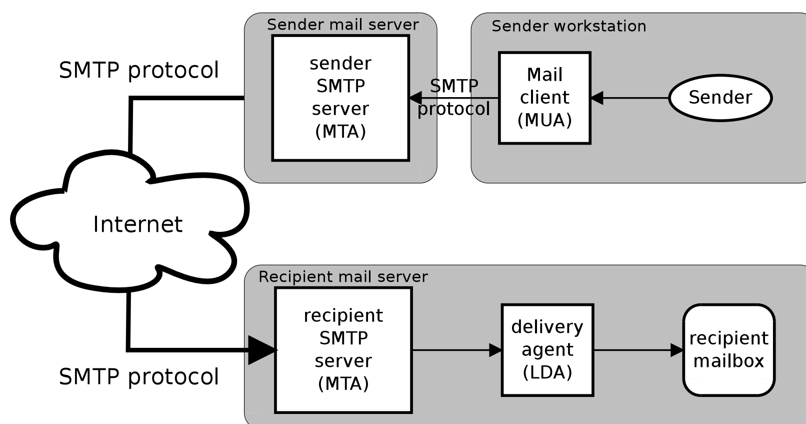


Figura 8.3: Schema del funzionamento della posta elettronica nel caso di invio da una workstation.

In questo caso, illustrato in fig. 8.3, è compito del client di posta essere in grado di parlare opportunamente con il server di posta per richiedere l'invio. In genere viene usato lo stesso protocollo SMTP utilizzato nella comunicazione fra server di posta su Internet, solo che di norma in questo caso il server a cui ci si rivolge è posto all'interno della stessa rete della macchina da cui si effettua la spedizione.

[34] l'*Unix to Unix CoPy* è un servizio di comunicazione che consentiva di inviare dati ed eseguire comandi su collegamenti temporanei (come linee telefoniche in *dial-up*); oggi è ancora utilizzabile essendo stato incapsulato su TCP/IP, ma sostanzialmente dismesso, anche se può presentare vantaggi non indifferenti rispetto a SMTP consentendo il trasferimento di più messaggi in blocco.

8.5.2 La posta elettronica sulla riga di comando

Come accennato in sez. 8.5.1 la posta elettronica può essere gestita anche soltanto localmente, il servizio infatti nasce ben prima di Internet e della possibilità di collegare in rete fra loro dei computer, come possibilità, presente sui sistemi multiutente come Unix, di mandarsi messaggi fra utenti diversi della stessa macchina.

Esistono pertanto una serie di programmi che consentono di leggere ed inviare posta dalla riga di comando, e la funzionalità di invio di messaggi di posta viene utilizzata da vari demoni di sistema, come `cron`, quando è necessario utilizzare una qualche forma di comunicazione asincrona con gli utenti, non potendo avere la certezza che questi siano collegati e contattabili direttamente.

Il programma deputato tradizionalmente alla lettura della posta elettronica è `mail`, che consente sia di leggere che di inviare posta elettronica. In realtà del programma esistono diverse versioni che replicano il comportamento del programma originale installato a partire dalla versione 1 dello Unix della AT&T, ed in genere su una distribuzione Linux si trova installato o la versione di BSD (che può essere invocato anche come `mailx`) o una versione alternativa derivata anch'essa dal comando di BSD chiamata *Heirloom-mailx*. Purtroppo le due versioni differiscono nell'uso di alcune opzioni più avanzate ed occorre tenere presente quale si sta usando, per entrambe comunque si può invocare il comando sia come `mail` che come `mailx`.

In generale il comando `mail` consente anche di leggere e scrivere la posta interattivamente sul terminale ma l'interfaccia è molto spartana e poco utilizzabile per cui per questo compito gli si preferiscono altri programmi. Se invocato senza opzioni o argomenti legge la posta (locale) dell'utente che lo avvia, ma si può richiedere la lettura della posta di un altro utente (posto che si abbia la capacità di farlo) specificandolo come argomento per l'opzione `-u`. Si può anche far leggere la posta da un file in formato *mbox* indicandone il pathname come argomento per l'opzione `-f`.

L'uso più comune del comando `mail` è però quello che lo vede utilizzato per inviare messaggi di posta all'interno di script o nell'invocazione di comandi sulla shell. La sintassi di base del comando infatti è di leggere il testo del messaggio da inviare dallo *standard input* prendendo l'indirizzo di posta del destinatario come argomento. Si può poi impostare l'oggetto del messaggio come parametro per l'opzione `-s`, ma si ricordi, se questo contiene degli spazi, di indicarlo esplicitamente come stringa proteggendolo dalla interpretazione della shell. Un esempio di uso del comando potrebbe essere allora:

```
piccardi@hain:~$ echo prova | mail -s "Prova messaggio" piccardi@truelite.it
```

che invierà il messaggio "`prova`" al destinatario indicato usando come mittente l'utente corrente.

Si tenga presente che in questo caso l'indirizzo di posta del mittente viene costruito sulla base del nome utente di chi ha lanciato il comando, a cui verrà aggiunto il nome a dominio della macchina, e non è detto che questo sia un indirizzo valido al di là della macchina stessa. Con `heirloom-mailx` comunque si può usare l'opzione `-r` per indicare esplicitamente l'indirizzo di posta da utilizzare come mittente.

Il comando consente poi di specificare eventuali indirizzi in copia (CC), con una lista separata da virgole degli stessi passata come argomento dell'opzione `-c`. Per inviare il messaggio in copia nascosta (BCC) bisognerà invece passare la lista come argomento dell'opzione `-b`. Altre opzioni comuni del comando sono `-E`, che evita l'invio di un messaggio vuoto, e `-d` che fa stampare al comando una serie di informazioni di debug.

L'uso di ulteriori opzioni dipende dalla versione del comando che si sta utilizzando, ad esempio per `bsd-mailx` l'opzione `-a` consente di aggiungere intestazioni ad hoc per il messaggio che si invia, mentre `heirloom-mailx` viene usata per indicare un file da aggiungere come allegato al messaggio che si invia. Per i dettagli si rimanda alla lettura delle rispettive pagine di manuale.

8.5.3 Le funzionalità di compatibilità con *Sendmail*

Benché, come accennato in sez. 8.5.1, esistano diversi MTA in grado di gestire invio e recapito della posta, una serie di comandi di gestione e di file di configurazione di base derivano dalle scelte implementative realizzate da *Sendmail*, che a lungo è stato praticamente l'unico programma utilizzato per questo compito.

Ma anche se oggi *Sendmail* può essere tranquillamente sostituito da alternative più efficienti e sicure come *Postfix*, detti comandi e file di configurazione, e le funzionalità che essi forniscono, sono stati sostanzialmente standardizzati e restano praticamente gli stessi con qualunque MTA si stia utilizzando.

La prima di queste funzionalità è quella degli *alias* di posta elettronica: è possibile cioè definire degli *pseudonimi* e fare in modo che la posta ad essi inviata venga reindirizzata altrove. Questo fa sì che ad esempio si possano usare diversi nomi per identificare un utente all'interno di un dominio, recapitando tutti i messaggi sempre alla stessa persona. Oppure si può recapitare ad una o più persone la posta inviata a indirizzi di posta associati a dei servizi (come `postmaster` o `webmaster`) che non corrispondono ad un utente reale, o la posta inviata a `root` che ovviamente non deve mai essere letta direttamente usando l'utente di amministrazione.

Questa funzionalità viene gestita in maniera generica da tutti gli MTA attraverso il file `/etc/aliases`, un estratto del quale, preso da una Debian Squeeze, è il seguente:

/etc/aliases
```
# /etc/aliases
mailer-daemon: postmaster
postmaster: root
nobody: root
hostmaster: root
usenet: root
news: root
webmaster: root
www: root
ftp: root
abuse: root
noc: root
security: root
root: piccardi
```

Nel file si mantengono le corrispondenze fra un *alias*, che deve iniziare sul primo carattere di una riga ed essere terminato dal carattere "`:`", e la destinazione cui si vuole la posta ad esso inviata arrivi, che deve essere indicata nella parte di riga seguente. Il file segue la solita convenzione dei file di configurazione di Unix di ignorare righe vuote ed inizianti per "`#`", inoltre una riga iniziante per spazi viene considerata una continuazione della precedente.

Si noti come un *alias* può indicare sia il nome di un utente realmente esistente sulla macchina di cui si vuole girare la posta ad un altro, come `root`, sia utenti fittizi, come `postmaster`. Si noti

anche come sia possibile usare come destinazione un altro *alias*, come nell'esempio illustrato dove si redirige `mailer-daemon` su `postmaster`, questi su `root` e quest'ultimo su `piccardi`. In questo caso non è necessario neanche scrivere le corrispondenze in ordine sequenziale.

Se allora un utente desidera essere contattato via posta con altri nomi oltre al suo username, ad esempio se vuole ricevere la posta come `simone.piccardi` o `s.piccardi`, si potranno aggiungere ad `/etc/aliases` le righe:

```
/etc/aliases
s.piccardi: piccardi
simone.piccardi: piccardi
```

Se invece si vuole smistare la posta inviata ad un certo indirizzo verso più destinatari, basterà inserirne la lista al posto del un singolo nome utente presente negli esempi finora illustrati. Questa lista dovrà essere indicata con un elenco, separato da spazi, sia di nomi di utenti locali sia di indirizzi di posta remoti. Per cui se ad esempio si vuole che la posta inviata all'amministratore della macchina oltre che all'utente `piccardi` venga inviata anche all'indirizzo remoto `reports@truelite.it`, si dovrà modificare l'ultima riga dell'esempio iniziale in:

```
/etc/aliases
root: piccardi reports@truelite.it
```

Una ulteriore funzionalità è quella che consente di inviare i messaggi di posta elettronica ad uno specifico programma che si incarica di gestirle, ad esempio una modalità di configurazione del gestore di liste "*Mailman*" è quella che prevede che si definiscano una serie di alias nella forma:

```
/etc/aliases
## mailman mailing list
mailman:              "|/var/lib/mailman/mail/mailman post mailman"
mailman-admin:        "|/var/lib/mailman/mail/mailman admin mailman"
mailman-bounces:      "|/var/lib/mailman/mail/mailman bounces mailman"
mailman-confirm:      "|/var/lib/mailman/mail/mailman confirm mailman"
mailman-join:         "|/var/lib/mailman/mail/mailman join mailman"
mailman-leave:        "|/var/lib/mailman/mail/mailman leave mailman"
mailman-owner:        "|/var/lib/mailman/mail/mailman owner mailman"
mailman-request:      "|/var/lib/mailman/mail/mailman request mailman"
mailman-subscribe:    "|/var/lib/mailman/mail/mailman subscribe mailman"
mailman-unsubscribe:  "|/var/lib/mailman/mail/mailman unsubscribe mailman"
```

il cui risultato è far si che le email inviate agli indirizzi indicati vengano processate inviandole, tramite l'uso del carattere "`|`", al programma `/var/lib/mailman/mail/mailman` che si incaricherà di gestirle in modo opportuno.

Una volta modificato il file `/etc/aliases` occorre comunque segnalarne il cambiamento in maniera esplicita, perché il server di posta possa utilizzare le nuove impostazioni. Per questo è previsto l'uso dell'apposito comando `newaliases`, che non prevede nessun argomento o opzione, ed effettua tutte le operazioni necessarie che potranno variare a seconda dell'MTA che si sta utilizzando.

Oltre agli *alias*, che possono essere gestiti soltanto dall'amministratore del servizio di posta elettronica, un'altra funzionalità derivata da *Sendmail* è quella che consente ai singoli utenti di

redirigere la propria posta verso un altro indirizzo a loro scelta. Questo può essere ottenuto scrivendo detto indirizzo dentro il file `.forward` posto nella propria home directory.

In realtà, come per gli *alias*, anche in questo caso si possono specificare più indirizzi come lista separata da spazi ed in presenza di questo file il server, all'atto di recapitare la posta all'utente, ne leggerà il contenuto e la reinvierà automaticamente agli indirizzi ivi specificati.

Un altro comando generico usato nella gestione della posta è `mailq`, che stampa la coda dei messaggi in attesa di spedizione. L'uscita del comando dipende dall'MTA che si sta utilizzando, ma il risultato è quello di mostrare un elenco di messaggi con le relative informazioni. Ad esempio con *Postfix* si otterrà qualcosa del tipo:

```
root@hain:~# mailq
-Queue ID- --Size-- ----Arrival Time---- -Sender/Recipient-------
494FD52A00C     1886 Wed Aug 22 23:43:34  pinco@truelite.it
        (connect to friends.filly.com[76.74.177.175]:25: Connection timed out)
                                         nobody@friends.filly.com

D5FE2F82A6     5593 Wed Aug 22 16:07:27  MAILER-DAEMON
      (connect to smtp.anbid.com.br[200.186.108.102]:25: Connection timed out)
                                         magnumsu00@anbid.com.br

75146F82A5     5131 Tue Aug 21 23:06:13  MAILER-DAEMON
                  (connect to dhl.ie[165.72.192.235]:25: Connection timed out)
                                         specializationbhk@dhl.ie

-- 13 Kbytes in 3 Requests.
```

che mostra dei messaggi (in questo caso notifiche di errore) che non sono stati recapitati per impossibilità di connettersi al server del destinatario.

In genere l'uso di `mailq` è a scopo diagnostico per capire se ci sono problemi nella spedizione della posta elettronica da parte del proprio server, ad esempio a causa di un eventuale crollo della connessione di rete verso l'esterno. La lista dei messaggi e le relative indicazioni permettono di capire lo stato del proprio sistema di posta, il comando non prende argomenti e non prevede l'uso di nessuna opzione.

Sia `mailq` che `newaliases` vengono in genere realizzati come link simbolici al programma `sendmail`, che nel caso di *Sendmail* è anche il binario del server stesso, mentre per altri MTA costituisce solo il programma che consente di inviare istruzioni al server usando l'interfaccia di compatibilità con *Sendmail*.

In particolare i due comandi citati sono equivalenti all'uso di `sendmail` con l'opzione `-I` (per `newaliases`) e `-bp` (per `mailq`). Oltre a queste due opzioni, che però non vengono usate esplicitamente, un'altra opzione molto usata è `-q`, che serve per forzare il server di posta a tentare il reinvio immediato di tutti i messaggi rimasti in coda (quella visualizzata con `mailq`).

In genere `sendmail -q` è il comando che si esegue sempre quando la connessione si riattiva dopo un periodo di inattività per evitare che il reinvio dei messaggi venga effettuato con i tempi standard (che in genere prevedono attese di qualche ora) che sono usati per i problemi di connessione. In condizioni normali infatti, come accennavamo in sez. 8.5.1, è previsto che il server destinatario di un messaggio possa non essere raggiungibile, e si debba ritentare in un secondo tempo. Quando si sa, come nel caso citato, che le condizioni di irraggiungibilità non sussistono più, si può richiedere di ritentare l'invio immediatamente.

Oltre a quelle citate il comando `sendmail` ha una lunga serie di altre opzioni molte delle quali attengono al funzionamento del *Sendmail* originale e o vengono ignorate o, se possibile ed applicabile, vengono tradotte in istruzioni equivalenti per il proprio MTA. La trattazione delle stesse va al di là di quanto possa interessare affrontare in questa sede e si rimandano gli interessati alla lettura della pagina di manuale.

Capitolo 9

Il servizio DNS

9.1 Il funzionamento del servizio DNS

In questa prima sezione ci occuperemo di illustrare i principali concetti ed introdurre i meccanismi di funzionamento del servizio DNS, descrivendone le caratteristiche generali ed una serie di programmi diagnostici usati appunto per eseguire interrogazioni su questo servizio.

9.1.1 Introduzione

Il DNS è uno dei protocolli fondamentali per il funzionamento di Internet, consentendo di associare dei nomi simbolici agli indirizzi IP, e come già accennato in sez. 7.4 si tratta in sostanza di un meccanismo che consente di realizzare un enorme database distribuito, i cui dati sono gestiti da un gran numero di *nameserver* (i server che rispondono alle richieste del protocollo).

Alle origini, quando Internet era piccola e le macchine erano poche, la risoluzione dei nomi era fatta scaricando su ogni macchina un singolo file (in sostanza `/etc/hosts`) che conteneva *tutte* le corrispondenze fra indirizzi IP e nomi. Col crescere della rete questo approccio è rapidamente divenuto insostenibile, non tanto e non solo per le dimensioni crescenti del file che doveva essere riscaricato ad ogni aggiornamento, quanto per la quasi impossibilità di mantenere aggiornate in maniera puntuale le corrispondenze stesse. Per questo è stato creato il protocollo del DNS, il cui scopo principale era sia quello di poter distribuire il compito di mantenere aggiornata l'associazione fra nomi simbolici e indirizzi IP che quello di poter eseguire la risoluzione in maniera veloce ed efficiente tramite un apposito servizio.

Il funzionamento del DNS è basato su una suddivisione gerarchica dello spazio dei nomi nei cosiddetti *domini*, organizzati in *livelli* con una struttura ad albero. I nomi vengono indicati in maniera generica da stringhe separate da punti che costituiscono quello che viene chiamato il cosiddetto *nome a dominio completo*, o FQDN (acronimo di *Fully Qualified Domain Name*). Si hanno così i domini di primo livello (i vari `.com`, `.org`, `.it`, ecc.) quelli di secondo livello (ad esempio `debian.org`, `gnulinux.it`, `truelite.it`) e così via, come illustrato in fig. 9.1.

Per decentralizzare l'aggiornamento delle informazioni si sfrutta un meccanismo chiamato *delegazione*. In generale il protocollo prevede che per ciascun dominio di un certo livello ci debba essere un server DNS responsabile della risoluzione di tutti i nomi del ramo di albero ad

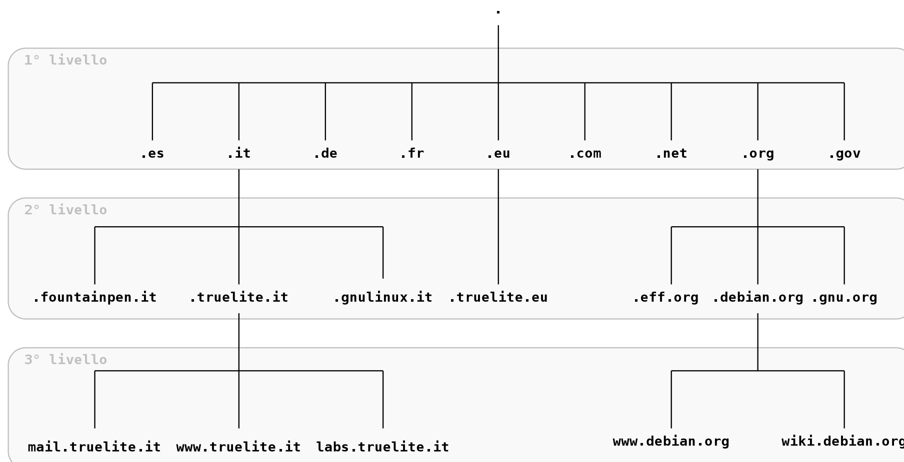


Figura 9.1: Schema dell'organizzazione gerarchica dei *nomi a dominio*.

esso sottostante, che per questo viene detto *autoritativo*. Grazie alla struttura gerarchica dei nomi è però possibile che il server responsabile per un dominio di un certo livello possa *delegare* altri server per la risoluzione dei domini dei livelli successivi, che a quel punto diventeranno *autoritativi* per tali sotto-domini.

In questo modo le richieste di risoluzione possono essere passate attraverso una gerarchia di server in cui, in corrispondenza di ciascun livello su cui si può dividere un nome a dominio, ci deve essere un server DNS che se anche non è in grado di rispondere direttamente alla richiesta sa però qual'è il server del livello successivo a cui inoltrarla, così da poter proseguire fino a che non si incontra il server DNS autoritativo che conosce la risposta finale.

Perché il sistema possa funzionare occorre però anche poter distribuire il carico delle richieste di risoluzione; se ciascuna macchina che si affaccia su Internet dovesse effettuare tutte le sue richieste direttamente ai server autoritativi si genererebbe infatti una enorme quantità di traffico. Per questo motivo come visto in sez. 7.4.1, il sistema del *resolver* consente di passare le richieste di risoluzione ad un server DNS esterno che si incaricherà di eseguire le ricerche necessarie, fornendo poi solo la risposta finale.

Perché questo funzioni entra in gioco un secondo meccanismo, denominato *ricorsione*, che è in genere quello che viene utilizzato dai server DNS a cui si rivolge il *resolver* (come quelli del proprio provider, che per questo sono detti *ricorsivi*).[1] La ricerca di una risoluzione infatti avviene *ricorsivamente*, con una scansione dell'albero dei domini come illustrato in fig. 9.2.

Se allora per esempio viene richiesta la risoluzione del nome `www.truelite.it` ad un DNS ricorsivo, questi all'inizio contatterà un *root DNS* per sapere chi è il server autoritativo per il dominio di primo livello `.it`. La lista degli indirizzi IP di questi DNS radice ovviamente deve essere nota a priori, altrimenti non sarebbe possibile iniziare la ricerca, ed è proprio per questo

[1] si tenga presente che, essendo questo un procedimento oneroso, si possono anche configurare server non ricorsivi, che forniscono solo risposte alle richieste per i quali sono autoritativi o che hanno in cache.

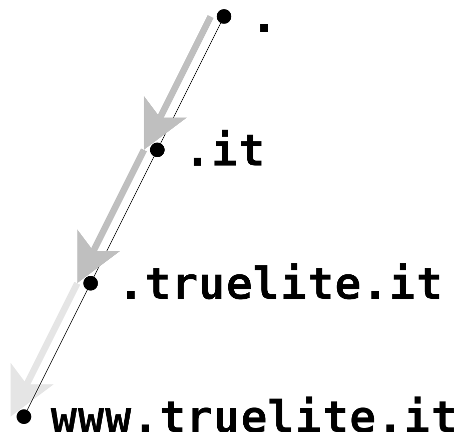


Figura 9.2: Schema della risoluzione ricorsiva di un nome a dominio.

che vengono chiamati "*root DNS*". La lista dei "*root DNS*" è distribuita pubblicamente, ed ogni server DNS ricorsivo deve averla all'interno della propria configurazione.

I domini di primo livello sono definiti a livello internazionale dalla cosiddetta *naming authority*, la ICANN (*Internet Corporation for Assigned Names and Numbers*), che gestisce i *root DNS* e delega i domini nazionali alle varie autorità locali. La lista dei DNS autoritativi per i domini di primo livello è mantenuta direttamente nei *root DNS* che hanno effettuato la relativa delegazione.

Nel caso del nostro esempio una volta che questi avranno indicato al richiedente chi è il server responsabile per i domini `.it` la scansione proseguirà ripetendo la richiesta per `.truelite.it` al DNS, uno di quelli del NIC italiano, appena trovato.[2] Essendo compito di questo server conoscere (e delegare) tutti i domini di secondo livello, sarà in grado di indicarvi qual'è il server DNS responsabile per `.truelite.it` a cui rivolgersi in ultima istanza per chiedere la risoluzione di `www.truelite.it`.

Un altro tassello del meccanismo è che in tutti questi passaggi il DNS ricorsivo che esegue la ricerca memorizzerà le varie informazioni ottenute a ogni passo nella sua cache per poterle riutilizzare immediatamente. Così si potrà evitare una ulteriore richiesta ai *root DNS* per i `.it` se si deve risolvere `.gnulinux.it`, oppure un'altra richiesta per `.truelite.it` se si deve risolvere `labs.truelite.it`, e si potrà rispondere direttamente senza ripetere nessuna ricerca ad una successiva richiesta di `www.truelite.it`. Questo introduce però anche una distinzione fra le risposte, che sono dette *autoritative* se provengono direttamente dal *nameserver* che ha l'autorità per quella richiesta e *non autoritative* se provengono dalla cache di un altro server.

Con queste modalità di funzionamento il DNS permette di distribuire in maniera efficace il compito di fornire le risposte e di mantenere aggiornata la corrispondenza fra nomi ed indirizzi IP, in quanto alla fine è il responsabile di ciascun dominio finale a dover mantenere un DNS che

[2] in realtà qui si è semplificato, a tutti i livelli i server autoritativi sono sempre più di uno e nelle risposte si ottiene una lista e non un singolo server, ma le richieste possono essere fatte ad uno qualunque ottenendo (configurazioni errate a parte) gli stessi dati.

contenga le informazioni relative alle sue macchine. Inoltre il meccanismo del mantenimento in cache dei risultati permette di aumentare l'efficienza della ricerca, evitando di ripetere più volte le stesse operazioni, riducendo il traffico sulla rete ed il carico sui server autoritativi.

Ovviamente la memorizzazione dei risultati ha anche un costo, in quanto se viene eseguito un cambiamento in una associazione fra nome a dominio e indirizzo IP questo non verrà visto da un server DNS fintanto che questi ha un'altra associazione valida nella propria cache. Per questo motivo tutte le informazioni del DNS sono corredate da un tempo di scadenza, denominato *time to live* (TTL), da scegliere opportunamente a seconda delle frequenze con cui queste possono cambiare, che permette, con tempi più o meno rapidi a seconda di quanto impostato,[3] di propagare i cambiamenti su tutta la rete.

Si tenga presente infine che il protocollo DNS permette di gestire vari tipi di informazione, finora infatti abbiamo parlato di risoluzione dei nomi a dominio in indirizzi IP, ma questi sono soltanto uno dei diversi tipi di dato che sono associabili ad un nome a dominio, e come vedremo più avanti il sistema consente di mantenerne diversi altri, come indirizzi IPv6, sinonimi, server di posta elettronica da contattare, dati relativi alla delegazione di un sottodominio, ecc.

9.1.2 I *Resorce Record* e le *zone di autorità*

Come accennato un server DNS è in grado di mantenere diversi tipi di informazione associati ad un dominio, che vengono chiamati, nel gergo del protocollo, *resource record* (spesso abbreviato nella documentazione in RR). Un *resource record* costituisce l'elemento di base dell'informazione mantenuta dal DNS, in sostanza una voce del database distribuito.

I *resource record* vengono classificati secondo la "*classe*" di dati a cui il loro contenuto fa riferimento e per il "*tipo*" dei dati stessi. In genere si parla di classe di indirizzi, in quanto i dati del DNS fanno comunque riferimento alla risoluzione di nomi. A ciascuna classe e a ciascun tipo viene in genere associata una stringa identificativa, ma oggi praticamente la sola classe in uso è quella degli indirizzi Internet, identificata dal nome `IN`.

Esistono invece molti *tipi* diversi di *resource record*, alcuni dei quali hanno un significato specifico rilevante per il funzionamento di alcuni protocolli: ad esempio i *record* `MX` vengono usati dai server di posta come previsto nel protocollo SMTP, mentre i *record* `NS` vengono usati dal DNS stesso per gestire la delegazione. Un elenco dei principali *tipi* è riportato in tab. 9.1, l'elenco completo viene mantenuto dalla IANA nella lista dei parametri del DNS, che è pubblicata sulla pagina web `http://www.iana.org/assignments/dns-parameters`.

I vari dati relativi ad un certo dominio vengono raccolti in quella che nel protocollo viene chiamata una *zona di autorità*; questa rappresenta in sostanza quella parte dello spazio dei nomi di dominio per i quali un singolo *nameserver* è responsabile di gestire le informazioni, sia fornendo direttamente le risposte, che delegando eventualmente ad un altro server la risoluzione di un dominio di livello inferiore (che a quel punto riceverà il compito di gestire la *zona di autorità* ad esso relativa).

In ogni caso il server al quale sia stata delegata una *zona di autorità* deve fornire le risposte, che in questo caso saranno quelle autoritative, per qualunque nome all'interno di quel dominio; non si può cioè delegare la risoluzione dei nomi relativi al dominio sul proprio livello, ma solo

[3] e sempre che i server DNS siano configurati correttamente per onorarle, cosa che non avviene sempre, dato che alcuni provider preferiscono risparmiare sulla banda e mantenere il più a lungo possibile le informazioni in cache ed i loro server DNS ignorano le indicazioni di scadenza fornite.

Tipo	Descrizione
`A`	corrispondenza fra nome a dominio e indirizzo IP.
`NS`	*nameserver* autoritativo per il dominio.
`CNAME`	nome alternativo (un alias ad un altro nome nel dominio).
`SOA`	dati dell'inizio della zona di autorità.
`PTR`	corrispondenza fra indirizzo IP e nome a dominio.
`MX`	server di posta per il dominio.
`TXT`	un commento o altro testo associato al nome a dominio.
`AAAA`	corrispondenza fra nome a dominio e indirizzo IPv6.
`SRV`	locazione di un servizio sul dominio.
`HINFO`	informazione su una stazione.
`KEY`	chiave crittografica.
`SIG`	firma digitale.

Tabella 9.1: Identificativo e relativa descrizione dei principali tipi di *resource record*.

quelli di un dominio del livello successivo. La delegazione è comunque facoltativa, si possono risolvere anche tutti gli eventuali sottodomini (o solo alcuni di essi) all'interno di un'unica *zona di autorità*, inserendo direttamente nella stessa anche i *record* per i nomi dei livelli successivi che non si vogliono delegare.

Il contenuto di una *zona di autorità* viene normalmente descritto tramite file di testo chiamati *file di zona* (su cui torneremo in sez. 9.2.4) in cui vengono raccolte le varie informazioni associate ai nomi in essa definiti, e le richieste di informazioni effettuate presso un server DNS sono in genere nella forma in cui si richiede il valore di un qualche *resource record* associato alla sua zona (vale a dire per un nome del relativo dominio).

9.1.3 I comandi `host` e `dig`

Per effettuare una richiesta di risoluzione di un indirizzo IP non è necessario in genere nessun programma specifico, basta un qualunque programma che usi il *resolver*, come avviene ad esempio quanto si invoca `ping` con un nome a dominio (di cui tra l'altro, come visto in sez. 7.6.1 stampa la risoluzione). In questo caso però si risolve solo l'indirizzo IP mentre come appena detto un server DNS mantiene anche informazioni diverse; per questo sono stati realizzati vari programmi in grado di eseguire interrogazioni generiche, in sostanza dei client per questo servizio.

Si tenga presente però che i suddetti programmi non usano il *resolver* ma parlano direttamente con un server DNS anche per effettuare la normale risoluzione dei nomi a dominio in indirizzi IP; per questo motivo non è detto che i risultati di una richiesta siano gli stessi che si potrebbero ottenere con gli ordinari client di rete (come `ping`) per i quali la risoluzione del nome potrebbe anche essere stata ottenuta da fonti alternative come `/etc/hosts` (si ricordi quanto detto in sez. 7.4) che in questo caso non sarebbero prese in considerazione.

Questo tipo di programmi può essere usato per verificare che l'impostazione della configurazione del *resolver* per l'uso dei DNS esterni funzioni correttamente, può accadere infatti che la rete sia configurata correttamente, ma non si riesca a fare nulla perché gli indirizzi simbolici non vengono risolti, ad esempio perché si è sbagliato ad indicare l'indirizzo del DNS in `resolv.conf`. L'uso più comune però è quello di controllare il funzionamento effettivo del proprio server DNS, e verificarne le informazioni.

Il primo comando che prenderemo in considerazione è `host`. Se invocato senza argomenti il comando stampa la sua sintassi, ma in genere lo si usa passandogli come argomento un nome a dominio che si vuole risolvere, di cui stampa la risoluzione con le informazioni ottenute dal server DNS; un esempio del suo utilizzo è:

```
piccardi@hain:~$ host www.penciclopedia.it
www.penciclopedia.it is an alias for gordon.penciclopedia.it.
gordon.penciclopedia.it has address 78.46.114.60
```

Il comando riporta in forma discorsiva che il nome `www.penciclopedia.it` è stato risolto come corrispondente all'IP `78.46.114.60`, ma la risposta ci mostra anche che in questo caso la corrispondenza del nome indicata è stata effettuata inizialmente con un *alias* ad un altro nome (cioè con un *record* di tipo `CNAME`, vedi tab. 9.1) che reinvia al nome principale della macchina, `gordon.penciclopedia.it`, che è quello per cui si ha invece la corrispondenza diretta con un indirizzo IP (cioè un *record* di tipo `A`).

Il comando funziona anche alla rovescia, si può cioè chiedere quale nome a dominio corrisponde ad un certo indirizzo IP (la cosiddetta *risoluzione inversa*) indicando quest'ultimo come argomento, con:

```
piccardi@hain:~$ host 78.46.114.60
60.114.46.78.in-addr.arpa domain name pointer gordon.truelite.it.
```

e si noti come in questo caso la risoluzione non corrisponda al nome associato in precedenza all'indirizzo IP. É infatti possibile che ci siano più nomi a dominio che corrispondono allo stesso indirizzo IP, ad esempio:

```
piccardi@hain:~$ host piccardi.gnulinux.it
piccardi.gnulinux.it has address 78.46.114.60
```

ci mostra ancora un altro nome a dominio associato allo stesso indirizzo; infatti nella risoluzione inversa (che si effettua con la richiesta di un *record* `PTR`) ad un indirizzo IP può essere associato un solo nome.

Il comando `host` permette di richiedere esplicitamente anche gli altri tipi di record usando l'opzione `-t`, che prende come parametro il tipo di record voluto indicato secondo i valori riportati in tab. 9.1. Ad esempio se vogliamo trovare il server incaricato della ricezione della posta per il dominio `truelite.it` possiamo usare:

```
piccardi@hain:~$ host -t MX truelite.it
truelite.it mail is handled by 30 mail3.truelite.it.
truelite.it mail is handled by 10 mail.truelite.it.
truelite.it mail is handled by 20 mail2.truelite.it.
```

che ci risponde restituendo i *record* `MX`. L'opzione supporta anche l'uso del valore speciale `ANY`, che richiede l'invio di tutti i *record* associati al nome.

Il comando effettua le sue richieste interrogando i server DNS elencati in `/etc/resolv.conf`, ma gli si può indicare una lista di server alternativi da utilizzare passando gli stessi come argomenti successivi al primo, e li si possono indicare anche per nome a dominio, ed in questo caso la risoluzione verrà effettuata dal *resolver*. Per le altre opzioni e tutti i dettagli si consulti la pagina di manuale.

Il secondo comando utilizzato per le interrogazioni ai DNS è `dig`; le sue funzionalità sono sostanzialmente le stesse di `host`, ma a differenza di questo è pensato come strumento di analisi

delle risposte dei server DNS, e riporta i risultati in un formato meno discorsivo ma molto più preciso, indicando tutti i dettagli relativi alle richieste fatte ed ai dati ottenuti, usando per i risultati lo stesso formato dei file di zona che vedremo in maggiore dettaglio in sez. 9.2.3.

Il comando richiede che si passi come argomento il nome a dominio di cui si vuole la risoluzione, ma in questo caso se non si indica nulla, il comando chiede la risoluzione per la radice dell'albero dei domini (cioè per il nome "`.`") stampando in sostanza l'elenco degli indirizzi dei *root server*.

Benché la lettura dell'output del comando sia più complessa, le informazioni sono molto più dettagliate e consentono di effettuare richieste in maniera molto più sofisticata, un esempio di risultato del comando è il seguente:

```
piccardi@hain:~$ dig piccardi.gnulinux.it

; <<>> DiG 9.7.0-P1 <<>> piccardi.gnulinux.it
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 38026
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;piccardi.gnulinux.it.          IN      A

;; ANSWER SECTION:
piccardi.gnulinux.it.   84656   IN      A       78.46.114.60

;; AUTHORITY SECTION:
gnulinux.it.            84657   IN      NS      ns.gnulinux.it.
gnulinux.it.            84657   IN      NS      ns2.gnulinux.it.

;; ADDITIONAL SECTION:
ns.gnulinux.it.         84657   IN      A       62.48.34.25
ns2.gnulinux.it.        84657   IN      A       217.133.96.194

;; Query time: 6 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
;; WHEN: Fri Mar 18 01:46:49 2011
;; MSG SIZE  rcvd: 121
```

L'output prevede una stampa iniziale dei dettagli della richiesta effettuata, e riporta (nella sezione etichettata `QUESTION`) i dettagli della domanda di risoluzione effettuata, a cui seguono una serie di sezioni che oltre alla risposta alla richiesta stessa (nella sezione `ANSWER`) riporta ulteriori dati relativi a chi è responsabile per la risoluzione (i *record* `NS` della sezione `AUTHORITY`) e eventuali dati addizionali (gli indirizzi dei suddetti DNS nella sezione `ADDITIONAL`). Infine si noti come in coda al comando siano riportate informazioni aggiuntive sul server contattato e sulla tempistica delle risposte.

Il comando consente di indicare come `host` la richiesta di tipi di record diversi con l'opzione `-t`, ma in generale questi (ed una eventuale classe diversa dal default di `IN`) possono essere indicati dagli argomenti successivi della riga di comando. Un esempio di invocazione di questo tipo è la seguente, che consente di ottenere la versione del server DNS che si sta usando, facendo riferimento alla classe di indirizzi `CH` che ormai è presente sono un casi come questi:

```
piccardi@hain:~$ dig version.bind CH TXT
```

```
; <<>> DiG 9.7.0-P1 <<>> version.bind CH TXT
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46522
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;version.bind.                  CH      TXT

;; ANSWER SECTION:
version.bind.           0       CH      TXT      "9.7.3"

;; AUTHORITY SECTION:
version.bind.           0       CH      NS      version.bind.

;; Query time: 4 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
;; WHEN: Fri Mar 18 02:10:11 2011
;; MSG SIZE  rcvd: 63
```

Come per `host` il programma di default interroga i server DNS elencati in `/etc/resolv.conf`, ma si può indicare esplicitamente a quale server rivolgersi passandogli un argomento nella forma `@server`. Inoltre `dig` può essere usato in modalità non interattiva con l'opzione `-f`, che permette di effettuare le ricerche leggendo i comandi da un file, passato come argomento dell'opzione.

Occorre inoltre una accortezza quando si vuole effettuare una risoluzione inversa, il comando infatti, a differenza di `host`, tratta sempre il suo primo argomento come un nome a dominio, anche se si scrive un indirizzo IP, se si vuole che venga chiesto il record `PTR` per quell'indirizzo occorre dirlo esplicitamente specificando l'opzione `-x`.

9.2 La gestione di un server DNS

In questa sezione prenderemo in esame la gestione di un server DNS, concentrandoci in particolare su una implementazione particolare, quella di *BIND*, che detiene comunque il quasi monopolio delle installazioni dei server DNS su Internet.

9.2.1 Il server *BIND*

Essendo un protocollo implementato a livello di applicazione, il servizio del DNS viene fornito attraverso un opportuno demone. A livello internazionale (si ricordi quanto detto in sez. 7.2.4) è stato assegnato al protocollo la porta 53 (sia UDP che TCP) per rispondere a delle interrogazioni. Il server DNS più diffuso è *BIND*, anche se il nome del programma eseguito è `named`. Il programma è parte del pacchetto scritto originariamente da Paul Vixie, e mantenuto dall'Internet Software Consortium, che attualmente è giunto alla versione 9.

L'installazione del programma da pacchetto è prevista da tutte le maggiori distribuzioni,[4] e una volta installato viene lanciato direttamente negli script di avvio. Inoltre in genere si prevede che al programma venga dedicato un utente apposito, in modo da poterlo far eseguire con

[4] nel caso di Debian il pacchetto è `bind9`, nel caso di RedHat/CentOS è semplicemente `bind`.

privilegi ridotti. Inoltre con la versione 9 è fornito anche un apposito programma di controllo, `rndc`, che consente di eseguire, oltre alle classiche azioni di avvio, ripartenza e rilettura delle configurazioni, anche operazioni più avanzate, come il controllo delle zone gestite dal server. Se invocato senza argomenti stampa una lista di tutti i comandi disponibili, di cui si sono riportati i più significativi in tab. 9.2, altrimenti il comando deve essere indicato come primo argomento.

Comando	**Descrizione**
`reload`	forza la rilettura della configurazione e dei file di zona, si può forzare la rilettura di una specifica zona passata come secondo argomento.
`retransfer`	forza il trasferimento della zona, senza controllare se è cambiato il numero di versione.
`stats`	stampa le statistiche di uso del server.
`stop`	ferma il servizio.
`status`	stampa lo stato del server.

Tabella 9.2: I principali comandi utilizzabili con `rndc`.

Come accennato in genere il server `named` viene lanciato direttamente dagli script di avvio, ma lo si può anche avviare direttamente, qualora ad esempio si debbano effettuare delle prove. In tal caso si usa in generale l'opzione `-f` per mantenerlo associato al terminale, si poi se ne vogliono anche vede i messaggi si può usare invece `-g` che inoltre forza la scrittura di tutti i messaggi di log sullo *standard error*. Questi ultimi vengono controllati dall'opzione `-d` seguita da un valore numerico che indica il livello di debug. Si può poi specificare un file di configurazione alternativo al default con l'opzione `-c`.

Di norma il programma viene lanciato sempre usando l'opzione `-u`, che permette di indicare un utente dedicato, senza privilegi speciali, per conto del quale il programma sarà eseguito dopo aver effettuato l'inizializzazione del server per le normali operazioni del DNS.[5] L'utente usato dipende in genere dalla distribuzione e dall'installazione che si è effettuata. Per le altre opzioni si rimanda al solito alla pagina di manuale.

9.2.2 Il file `named.conf`

Il file di configurazione principale di `named` è `named.conf`, che di solito viene installato a seconda delle distribuzioni o sotto `/etc/named` o sotto `/etc/bind`, se non direttamente sotto `/etc`. Il file ha una sintassi abbastanza complessa e simile a quella dei programmi in C; da un punto di vista puramente "*grammaticale*" esso consiste di una serie di direttive, alcune delle quali a loro volta possono contenere blocchi di ulteriori direttive racchiusi fra parentesi graffe.

Ogni direttiva deve essere terminata con il carattere "`;`" che ne indica la conclusione, pertanto si possono scrivere le impostazioni di una direttiva su più righe senza nessuna sintassi particolare. Se la direttiva prevede un valore singolo questo deve essere specificato di seguito all'identificativo della direttiva separato da spaziatura, se la direttiva prevede valori multipli questi devono essere indicati all'interno di un blocco separati dal solito "`;`" di terminazione. Si tenga presente che detti valori possono essere a loro volta altre direttive che richiedono un blocco e così via.

Ad ogni parentesi graffa aperta, che indica l'inizio di un blocco, deve corrispondere una parentesi graffa chiusa e il "`;`" di terminazione che ne indicano la fine. Alcune direttive che

[5] dovendo porsi in ascolto sulla porta 53, che è una delle porte riservate, è necessario avere inizialmente i privilegi di amministratore che nel corso delle ulteriori operazioni non sono più necessari.

richiedono un blocco possono richiedere un ulteriore argomento singolo da indicare prima di questo (nel caso separato da spazi). I commenti possono essere introdotti dall'usuale carattere "`#`", ma viene utilizzata anche la sintassi del C e del C++, con blocchi delimitati da `/* */` o inizianti per `//`.

In realtà la situazione è più complessa in quanto anche se la sintassi utilizzata è la stessa, alcune direttive sono più importanti ed hanno un ruolo diverso dalle altre, dato che vanno a costituire la struttura di `named.conf`. Queste sono quelle che vengono utilizzate per impostare le principali funzionalità del programma, ed implicano sempre la presenza di un blocco contenente le ulteriori direttive secondarie che specificano dette impostazioni.

Dato il ruolo ed il significato completamente diverso parlare genericamente di direttive rischia di generare confusione, per cui d'ora in poi chiameremo sempre "*dichiarazioni*" queste direttive fondamentali, mentre chiameremo "*istruzioni*" tutte le altre direttive secondarie usate all'interno dei blocchi definiti dalle dichiarazioni, che configurano le funzionalità di queste ultime. Si tenga presente che alcune istruzioni possono essere a loro volta essere costituite con blocchi di altre istruzioni.

L'unica direttiva che non rientra in questa suddivisione, e che pertanto continuiamo a chiamare direttiva, è `include` che consente semplicemente di includere il contenuto di un altro file come se lo si fosse scritto direttamente. In questo modo diventa possibile dividere le varie configurazioni (ad esempio le zone per diversi domini, o le configurazioni generiche) e mantenere separatamente le varie parti.

Una lista delle principali dichiarazioni che si possono utilizzare all'interno di `named.conf` del loro significato generico è la seguente:

`acl` definisce una *Access Control List* per raggruppare macchine o utenti (quest'ultimi identificati da chiavi definite da `key`) a cui fare riferimento all'interno di altre dichiarazioni.

`controls` definisce i canali di controllo del demone usati per l'amministrazione remota (gli accessi per le operazioni effettuate dal comando `rndc`). Può essere omessa ed in tal caso si applica comunque una configurazione di default che prevede l'accesso solo da `localhost` sulla porta 953, utilizzando la chiave posta nel file `rndc.key`.

`key` definisce una chiave crittografica condivisa,[6] usata per identificare ed autenticare le comunicazioni con il server (come i canali di controllo definiti con `controls`). Per l'uso di `rndc` è previsto l'uso della dichiarazione in un file separato `rndc.key` (nella stessa directory di `named.conf` e con la stessa sintassi) che in genere viene creato automaticamente all'installazione del pacchetto.

`logging` definisce le modalità con cui le varie informazioni vengono inviate sui file di log o al sistema del *syslog*.

`options` permette di impostare alcune proprietà generali del server ed i default per le altre direttive. Conviene inserirla in un file a parte (ad esempio in Debian è mantenuta `/etc/bind/named.conf.options`) da includere con `include`.

[6] *BIND* supporta solo crittografia a chiave simmetrica (vedi sez. 1.2.2 di [SGL].

view definisca una *vista* del DNS, che consente di modificare risposte e comportamento del server a seconda degli indirizzi da cui arrivano le richieste.

zone definisce una *zona* del DNS e le modalità con cui vengono utilizzate le relative informazioni. Una *zona* serve ad identificare una parte di nomi di dominio per il quale il server deve eseguire delle azioni specifiche. Richiede che si indichi come argomento la stringa del dominio a cui la zona si applica.

Si tenga presente che le due dichiarazioni `logging` e `options` possono comparire soltanto una volta. La lista completa delle dichiarazioni, con un sommario delle relative istruzioni possono essere trovati nella pagina di manuale accessibile con `man named.conf`. Per una descrizione completa occorre fare riferimento alla documentazione di *BIND* che è molto estesa, e può essere consultata in linea all'indirizzo `http://www.isc.org/software/bind/documentation`, dove sono rese disponibili anche delle versioni in PDF del manuale di riferimento del programma.

Un esempio dell'inizio del file `named.conf`, ripreso dal file installato dal pacchetto `bind` su una distribuzione Debian Squeeze, è riportato di seguito:

named.conf
```
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local

include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";
```

in cui semplicemente vengono inclusi una serie di altri file.

9.2.3 La configurazione base di *BIND*

Le modalità di utilizzo di un server DNS possono essere le più svariate, affronteremo qui solo quelle relative al suo uso all'interno di una rete locale. La modalità più semplice è quella del cosiddetto *caching DNS*, cioè di usare il server come una specie di memoria tampone per evitare di ripetere ogni volta le richieste di risoluzione dei nomi su Internet presso il DNS del proprio provider. In genere installando il pacchetto di *BIND* fornito da una distribuzione si ottiene appunto una configurazione di questo tipo.

In questo caso quello che si troverà nei file di configurazione sono una serie di configurazioni di base; le opzioni generali per il server vengono sempre effettuate con la dichiarazione `options`, e si avrà pertanto una sezione che conterrà almeno qualcosa del tipo di:

named.conf
```
options {
        directory "/var/cache/bind";
        ...
};
```

in cui viene usata la ulteriore istruzione `directory` per indicare quale è la directory nella quale il server andrà a cercare i suoi file; anche questa può variare a seconda della distribuzione.

Il primo passo da fare è quello di far sapere al server dove può trovare i *root DNS*; questo viene fatto inserendo in `named.conf` una zona apposita, usando una sezione del tipo di:

named.conf
```
zone "." {
        type hint;
        file "/etc/bind/db.root";
};
```

in questo caso il comando definisce una zona per la radice (che nel sistema dei nomi di dominio è indicata da un `"."`). I file preimpostati nell'installazione standard già prevedono una sezione di questo tipo.

Le zone possono essere di vari tipi, come specificato dall'istruzione `type`, in questo caso si ha una zona di tipo `hint` che dice semplicemente a chi richiedere le informazioni relative a quella zona. L'istruzione `file` specifica il file da cui prendere le informazioni, questo può essere indicato in forma assoluta, come nell'esempio, o in forma relativa rispetto alla directory definita dall'istruzione `directory` inserita nella dichiarazione `options`. In questo caso il file è `/etc/bind/db.root` che contiene l'elenco dei *root DNS*, questo può essere ottenuto con il comando `dig` (vedi sez. 9.1.3) o scaricato direttamente all'indirizzo `ftp://ftp.internic.net/domain/named.root`.

La sintassi generale di una dichiarazione `zone` prevede che essa sia seguita, fra virgolette (`"`), dal nome della zona a cui si fa riferimento e da un blocco di istruzioni che specificano le caratteristiche della zona. In generale, oltre alla zona per i *root DNS*, quando si installa il pacchetto di `bind`, i file di configurazione sono preimpostati per la risoluzione del `localhost`. Vedremo i dettagli al riguardo in sez. 9.2.4, quando tratteremo la configurazione del *nameserver* per la risoluzione di un dominio locale.

Di norma l'installazione standard di *BIND* prevede che tutto questo venga preimpostato, per cui se se ne vuole verificare il funzionamento, basterà fare una richiesta per un indirizzo fino ad allora non usato. Allora ad esempio la prima volta avremo che:

```
piccardi@hain:~$ dig gapil.gnulinux.it
; <<>> DiG 9.7.3 <<>> gapil.gnulinux.it
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29931
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;gapil.gnulinux.it.             IN      A

;; ANSWER SECTION:
gapil.gnulinux.it.      600     IN      A       78.46.114.62

;; AUTHORITY SECTION:
gnulinux.it.            600     IN      NS      ns.gnulinux.it.
gnulinux.it.            600     IN      NS      ns2.gnulinux.it.

;; ADDITIONAL SECTION:
ns.gnulinux.it.         600     IN      A       62.48.34.25
```

```
ns2.gnulinux.it.        600     IN      A       217.133.96.194

;; Query time: 233 msec
;; SERVER: 62.48.34.25#53(62.48.34.25)
;; WHEN: Fri Aug 24 16:33:15 2012
;; MSG SIZE  rcvd: 118
```

in cui la richiesta effettuata viene soddisfatta in 233 ms, ma se subito dopo si ripete una seconda interrogazione otterremo:

```
piccardi@hain:~$ dig gapil.gnulinux.it
; <<>> DiG 9.7.3 <<>> gapil.gnulinux.it
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8480
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;gapil.gnulinux.it.             IN      A

;; ANSWER SECTION:
gapil.gnulinux.it.      600     IN      A       78.46.114.62

;; AUTHORITY SECTION:
gnulinux.it.            600     IN      NS      ns.gnulinux.it.
gnulinux.it.            600     IN      NS      ns2.gnulinux.it.

;; ADDITIONAL SECTION:
ns.gnulinux.it.         600     IN      A       62.48.34.25
ns2.gnulinux.it.        600     IN      A       217.133.96.194

;; Query time: 1 msec
;; SERVER: 62.48.34.25#53(62.48.34.25)
;; WHEN: Fri Aug 24 16:33:17 2012
;; MSG SIZE  rcvd: 118
```

con i tempi che si riducono drasticamente a 1 ms, che ci mostra nella maniera più evidente che il nostro *caching nameserver* sta funzionando.

Con questa impostazione però è il nostro server che si incarica di effettuare il procedimento di scansione ricorsiva che abbiamo illustrato in sez. 9.1.1. Dato che questo comporta vari passaggi possiamo limitare ulteriormente le richieste che escono dalla nostra rete usando il *nameserver* del provider per effettuare la scansione ricorsiva per conto nostro e farci dare solo il risultato finale. Questo si fa aggiungendo alla dichiarazione `options` le specifiche per l'uso di altri server con l'istruzione `forwarders`, con una sezione del tipo:

named.conf
```
options {
        ...
        forwarders {
                213.234.128.211;
                213.234.132.130;
        };
        ...
};
```

ed in questo caso ogni richiesta verrà inoltrata verso i server DNS dichiarati nell'istruzione `forwarders`, e si otterranno i solo risultati finali.

9.2.4 La configurazione di un dominio locale.

Per illustrare con maggiore dettaglio il funzionamento di *BIND* vedremo in questa sezione come configurare un dominio locale, usabile ad esempio per la risoluzione dei nomi delle macchine all'interno di una rete locale. Per semplicità useremo un dominio completamente astratto, `earthsea.ea`. Per far questo dovremo definire la relativa zona, dovremo aggiungere quindi al file di configurazione le due dichiarazioni:

named.conf
```
zone "earthsea.ea" {
        type master;
        file "/etc/bind/db.earthsea";
};
zone "1.168.192.in-addr.arpa" {
        type master;
        file "/etc/bind/db.192.168.1";
};
```

In genere, per una migliore organizzazione, si tende a mantenere le zone locali su un file a parte, da includere dal file principale con la direttiva `include`, come nell'esempio mostrato in sez. 9.2.2. Si noti come si debbano definire due zone, entrambe di tipo `master`, una per la risoluzione diretta, relativa al dominio che abbiamo scelto, ed una per la risoluzione inversa degli indirizzi privati che stiamo usando (nel nostro caso si suppone di aver usato la rete `192.168.1.0`) sui nomi del suddetto dominio.

Per la risoluzione inversa si usa sempre il nome di dominio `in-addr.arpa`, riservato a questo scopo, utilizzando l'indirizzo IP del quale si vuole effettuare la risoluzione, espresso in forma *dotted decimal* in ordine rovesciato. Esso infatti viene interpretato comunque come un nome di dominio separato da punti, per cui il livello più generale di risoluzione è dato dalla parte più significativa del numero.

Inoltre in genere nella installazione standard di *BIND* è prevista la configurazione per la risoluzione in locale del `localhost`, il che di norma comporta la presenza dentro `named.conf` (o in uno dei file da esso inclusi) di due zone del tipo (si sono prese le impostazioni installare di default su una Debian Squeeze):

named.conf
```
zone "localhost" {
        type master;
        file "/etc/bind/db.local";
};

zone "127.in-addr.arpa" {
        type master;
        file "/etc/bind/db.127";
};
```

anche in questo caso la zona è di tipo `master` in quanto è sempre il nostro server ad essere il responsabile della risoluzione di questo nome.

Vediamo allora quale è il formato dei file di configurazione delle singole zone, che contengono i dati (i *resource record* di cui abbiamo già parlato in sez. 9.1.2) che il server deve fornire quando viene interrogato. Cominciamo col mostrare quello usato per il `localhost`, che illustra il contenuto generico di uno di questi file:

db.local
```
$TTL    604800
@       IN      SOA     localhost. root.localhost. (
                              1         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      localhost.
@       IN      A       127.0.0.1
```

I file possono contenere alcune direttive, identificate dall'uso del carattere "`$`" preposto ad una parola chiave. Le direttive disponibili sono standardizzate, come il formato dei file di zona; il loro elenco, con il rispettivo RFC di definizione, è riportato in tab. 9.3.

Direttiva	**Descrizione**
`$TTL`	imposta il valore di default del *time to live*, da indicare come argomento; usualmente viene specificato in secondi (RFC 2308).
`$ORIGIN`	imposta il dominio di base del file di zona (RFC 1035), in genere non la si usa essendo questo già determinato nella dichiarazione della zona.
`$INCLUDE`	consente di includere il contenuto di un altro file, da indicare come argomento (RFC 1035).

Tabella 9.3: Le direttive utilizzabili nei file di zona.

Normalmente viene usata solo `$TTL`, che permette di impostare, per tutti i record seguenti, il cosiddetto *time to live*, cioè il tempo massimo di validità del record nella cache di un *nameserver* (si ricordi quanto detto al proposito in sez. 9.1.1). La direttiva deve essere necessariamente usata all'inizio del file, prima della definizione di qualunque *resource record*. Nel nostro esempio il tempo è stato specificato in secondi, e corrisponde ad una settimana.

Il formato con cui può essere specificato un tempo all'interno di un file di zona, valido in generale e non solo per la direttiva `$TTL`, oltre alla indicazione numerica semplice che indica i secondi, prevede anche l'uso dei suffissi `H, D, W` per indicare rispettivamente ore, giorni e settimane.

Alle eventuali direttive iniziali seguono poi i vari *resource record*, in genere uno per riga, formati da campi separati da spazi o tabulazioni. Il primo campo deve iniziare dal primo carattere della riga, altrimenti verrà considerato vuoto. Se necessario si possono specificare i campi su più righe, racchiudendoli fra parentesi tonde. Infine si possono inserire commenti ovunque precedendoli con il carattere "`;`". Il formato dei vari record è simile, la sintassi generale è la seguente:

```
{<domain>|@|<blank>} [<ttl>] [<class>] <type> <rdata> [<comment>]
```

Il primo campo indica il nome di dominio che viene ricercato; esso può essere immesso esplicitamente, ma se il campo è vuoto (rappresentato nella sintassi come <blank>) viene usato il nome immesso nella prima riga precedente non vuota. Infine si può usare il carattere speciale "@", che viene automaticamente espanso nel nome della zona indicato dalla dichiarazione zone che fa riferimento al file. Questa è la cosiddetta "*origine*" che può essere modificata nel corso del file con la direttiva $ORIGIN.

Il campo ttl viene di norma omesso, in quanto di solito si usa la direttiva $TTL per specificarlo una volta per tutte all'inizio del file. Il campo <class> indica la classe dei dati del record; nel nostro caso usando indirizzi Internet sarà sempre IN per tutti i vari tipi di record, dato che le altre classi non vengono usate. Segue il campo <type> che definisce il tipo di record e deve essere sempre specificato usando uno dei valori di tab. 9.1, infine segue il campo (o i campi) <rdata> con i dati da inviare in risposta alle richieste relative al record, il cui formato dipende dal tipo dello stesso.

L'ordine in cui si dichiarano i record non è essenziale, ma è convenzione specificare sempre per primo un record di tipo SOA che dichiara che il nostro *nameserver* è *autoritativo* per quel dominio (di nuovo si ricordi quanto detto in sez. 9.1.1), dato che non ce ne può essere più di uno per file, questo significa anche la necessità di un file separato per ogni dominio che si vuole definire. In genere poi si prosegue con la dichiarazione dei record di tipo NS che definiscono i *nameserver* per il dominio in questione, a cui fanno seguito gli altri tipi di record.

La prima colonna di un record SOA è il nome del dominio per il quale si dichiara di avere autorità, nel caso in esempio si è usato il carattere speciale "@", altrimenti si sarebbe dovuto specificare il nome del dominio completo nella forma localhost., con il "." finale in quanto per il SOA i nomi vanno dichiarati in forma assoluta (facendo riferimento alla radice che è appunto ".") segue la classe, IN, ed il tipo, SOA. Nel caso di un record SOA i dati prevedono che si indichi il nome del server che farà da *nameserver* principale per il dominio e l'indirizzo e-mail del responsabile, con il "." al posto dell'ordinario "@" (nell'esempio riportato l'indirizzo è root@localhost).

Le restanti informazioni, inserite nell'esempio fra parentesi tonde per poterle dividere su più righe, sono usate nella replicazione dei dati fra *nameserver* primari e secondari. Questa è una funzionalità avanzata del protocollo che consente di mantenere delle repliche automatizzate dei dati di un *nameserver*, aggiornate secondo quanto stabilito da questi parametri, su un server di riserva (detto appunto secondario) che può essere utilizzato insieme al primario e rispondere se questo fallisce (tratteremo l'argomento in sez. 9.3.2). Di norma l'unica informazione da modificare è quella identificata nell'esempio dal commento serial che indica un numero seriale (crescente) da aggiornare tutte le volte che si effettua una modifica ai dati della zona.

Tornando all'esempio i due record successivi dichiarano rispettivamente con il record NS, la macchina nel dominio che fa da *nameserver*, che deve sempre essere un nome (anche fuori dal dominio) per il quale però esista una risoluzione. In questo nel caso il server DNS viene identificato dal nome del dominio stesso, ed il suo indirizzo IP viene specificato nella riga seguente con un record A.

Vista la particolarità del dominio localhost, che ci è servito per introdurre la sintassi dei file di zona, vediamo un esempio più significativo su come impostare un dominio per risolvere i nomi delle macchine di una rete locale. In questo caso si è scelto il dominio earthsea.ea per la rete 192.168.1.0/24 e si sono inseriti i relativi dati in un file db.earthsea, il cui contenuto è:

db.earthsea

```
$TTL 100000
@ IN SOA gont.earthsea.ea. piccardi.gont.earthsea.ea. (
                        2       ; serial
                        10800   ; refresh after 3 hour
                        3600    ; retry after 1 hour
                        604800  ; expire after 1 week
                        86400 ) ; minimum TTL 1 day
;
; Name server
;
@               IN      NS      gont.earthsea.ea.
;
; Mail server
;
                        MX      10 gont
                        MX      20 oppish
;
; Indirizzi
;
gont            IN      A       192.168.1.1
ns              IN      CNAME   gont
www             IN      CNAME   gont
hogen           IN      A       192.168.1.2
lorbaner        IN      A       192.168.1.17
karegoat        IN      A       192.168.1.19
oppish          IN      A       192.168.1.168
localhost       IN      A       127.0.0.1
```

Di nuovo si è cominciato con un record di tipo `SOA` che dichiara l'autorità per `earthsea.ea`, in questo caso il *nameserver* è la macchina `gont.earhsea.ea.`, anch'essa identificata dal suo nome di dominio assoluto. Si tenga conto che se non si usa il nome assoluto al nome viene sempre aggiunta l'*origine* della zona, per cui o si usa questa notazione o si specifica il *nameserver* semplicemente con `gont`. Per l'amministratore del dominio si è indicato che la posta venga inviata all'indirizzo `piccardi@gont.earthsea.ea`.

Il secondo record definisce il *nameserver*, cioè "`gont.earthsea.ea.`": si noti che di nuovo, come per il precedente, non lo si è definito con l'indirizzo IP ad esso corrispondente; tutto quello che occorre infatti è usare un nome che corrisponda ad un record di tipo `A`. Nel caso si è specificato un solo *nameserver*, ma se ne possono definire più di uno; l'informazione sarà passata nelle richieste di risoluzione, che potranno essere rivolte ad uno qualunque dei server della lista; occorrerà ovviamente che questi forniscano risposte coerenti, vedremo come si può ottenere questo in sez. 9.3.2.

Il record successivo, di tipo `MX`, serve a definire a quale macchina viene inviata la posta per il dominio `earthsea.ea`, di nuovo si è specificato `gont`, ma stavolta, per dare un esempio di come si possono scrivere le cose in maniera più compatta, non si è dichiarato il dominio (che essendo `blank` corrisponde a quello precedente, e cioè sempre `earthsea.ea`, si è tralasciata la classe `IN` che essendo quella di default può non essere specificata esplicitamente, ed infine si è indicato `gont` senza usare il nome assoluto. Il valore `10` inserito prima del nome indica la priorità del server di posta, è possibile infatti specificare più di un server di posta, con valori diversi, in questo modo verrà sempre contattato per primo quello con il valore più basso, passando ai successivi solo in

caso di fallimento dei precedenti. Si tenga presente che anche questi record è opportuno facciano riferimento a nomi associati a dei record di tipo `A`.

Infine si sono inseriti i vari record di tipo `A` contenenti gli indirizzi associati ai nomi, ed i record di tipo `CNAME` che definiscono dei nomi alternativi per le stesse macchine. Per quanto visto finora il formato di questi record è evidente.

Una volta definito il nostro dominio occorre anche definire la risoluzione inversa, questo è fatto tramite il file `db.192.168.1`, il cui contenuto è il seguente:

db.192.168.1
```
$TTL 100000
@ IN SOA gont.earthsea.ea. piccardi.gont.earthsea.ea. (
                        2       ; serial
                        10800   ; refresh after 3 hour
                        3600    ; retry after 1 hour
                        604800  ; expire after 1 week
                        86400 ) ; minimum TTL 1 day
;
; Name server
;
@               IN      NS      gont.earthsea.ea.
;
; Indirizzi
;
1               IN      PTR     gont.earthsea.ea.
2               IN      PTR     hogen.earthsea.ea.
17              IN      PTR     karegoat.earthsea.ea.
19              IN      PTR     lorbaner.earthsea.ea.
168             IN      PTR     oppish.earthsea.ea.
```

Di nuovo si comincia con un record di tipo `SOA`, sostanzialmente identico al precedente dato che *nameserver* e amministratore sono gli stessi. Si deve definire anche il *nameserver* per questo dominio inverso, che ovviamente è sempre `gont.earthsea.ea.`; qui specificare `gont` non avrebbe funzionato, essendo in una zona diversa, l'indirizzo assoluto invece funziona in quanto il dominio è stato già definito prima.

Seguono infine i record di tipo `PTR` che contengono il nome delle singole macchine, in questo caso però gli indirizzi devono corrispondere ad un solo nome, quello canonico, e contrariamente a prima non si possono associare più nomi allo stesso indirizzo. Anche se è possibile associare due nomi allo stesso IP molti sistemi non sono preparati a questa evenienza, che è bene pertanto escludere onde evitare comportamenti inattesi.

Completati i nostri file possiamo riavviare il servizio, e controllare i risultati. Al solito usiamo `dig` per interrogare il nostro server:

```
piccardi@gont~$ host -t any earthsea.ea
earthsea.ea has SOA record gont.earthsea.ea. piccardi.gont.earthsea.ea. 2 10800
 3600 604800 86400
earthsea.ea name server gont.earthsea.ea.
earthsea.ea mail is handled by 10 gont.earthsea.ea.
earthsea.ea mail is handled by 20 oppish.earthsea.ea.
```

ed otteniamo il risultato voluto, adesso possiamo provare a risolvere un'altra macchina:

```
piccardi@gont~$ host lorbaner.earthsea.ea
lorbaner.earthsea.ea has address 192.168.1.17
```

ed infine proviamo con la risoluzione inversa, ricordiamoci che in questo caso `dig` vuole l'opzione `-x`, ed otterremo:

```
piccardi@gont~$ host 192.168.1.17
17.1.168.192.in-addr.arpa domain name pointer karegoat.earthsea.ea.
```

e di nuovo è tutto a posto.

Si tenga presente che nonostante negli esempi fatti finora si sia fatto riferimento ad un dominio locale per una rete privata, le stesse direttive si possono applicare con indirizzi IP pubblici e nomi a dominio ufficiali, che in effetti vengono configurati esattamente in questo modo.

La differenza nel caso è che mentre per il dominio `earthsea.ea` non è possibile ottenere una delegazione, dato che i *Root DNS* non riconoscono il dominio di primo livello `.ea`, se avessimo usato `fountainpen.it` avremmo invece potuto ottenere la delegazione dal server del NIC italiano, ed i dati della nostra zona sarebbero diventati accessibili a chiunque li avesse richiesti con una interrogazione diretta verso questo dominio.

9.3 Configurazioni avanzate

Tratteremo in questa sezione alcune configurazioni più avanzate di *BIND*, come l'impostazione di eventuali secondari e la gestione delle delegazioni.

9.3.1 La delegazione di una zona

Come accennato una delle caratteristiche fondamentali del funzionamento del DNS è la capacità di *delegare* la risoluzione di parti dell'albero dei nomi ad altri server, in modo da distribuire sia il carico di rete che quello amministrativo.

Il problema deve ovviamente essere affrontato da due lati, quello del server DNS che esegue la delega, e quello del server che viene delegato. Per quest'ultimo come accennato la configurazione sarà una analoga a quella già illustrata in sez. 9.2.4, in cui invece di usare un dominio interno per la propria zona locale senza curarsi di cosa può stare al di sopra, si userà quello che viene delegato dal primo server. L'unica accortezza in questo caso è assicurarsi che all'interno di detto dominio vengano indicati dei record `NS` con una risoluzione coerente con quella indicata dal server che effettua la delega.

La *delegazione* pertanto attiene sostanzialmente alla configurazione del server responsabile per un certo dominio, che potrà effettuarla solo per un suo sottodominio. Si tenga presente che una volta che si delega un sottodominio i nomi al suo interno non possono più essere inseriti nello stesso file di zona in cui si è eseguita la delega, andando a far parte di una *zona di autorità* diversa. Fanno eccezione i cosiddetti *glue records*, che permettono di evitare una dipendenza circolare quando si delega una zona usando per i relativi DNS dei nomi in quella zona.

Ad esempio se `ns.truelite.it` è il server DNS della zona `truelite.it`, c'è il problema di come avere l'IP con cui contattarlo per ottenere la sua risoluzione dato che è lui che dovrebbe fornirlo. Questo problema viene superato fornendo al DNS che effettua la delegazione di `truelite.it` non solo il record `NS` per `truelite`, ma anche il record `A` per `ns.truelite.it` che per questo viene detto appunto *glue record*.

La configurazione effettiva della delegazione di una zona è pertanto da eseguire nel DNS che gestisce il dominio di livello superiore rispetto a quello che si intende delegare. Supponiamo

allora di avere il controllo del dominio `.trl` e voler delegare le zone `fi.trl` e `bz.trl`; in questo caso il file di zona relativo a `.trl` sarà qualcosa del tipo:

db.trl
```
$TTL    604800
@       IN      SOA     ns.trl. sistemi.truelite.it. (
                     2006061601         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@               IN      NS      ns
ns              IN      A       192.168.1.2
; delegazione
fi              IN      NS      ns.fi.trl.
ns.fi.trl.      IN      A       192.168.1.2
bz              IN      NS      ns.bz.trl.
ns.bz.trl.      IN      A       192.168.3.42
```

Come si può notare la delegazione si esegue semplicemente usando un record di tipo `NS` che fornisce l'indicazione di qual'è il DNS per quella zona. Siccome un record `NS` vuole come dato il nome a dominio del server DNS, in questo caso si sono specificati come *glue record* anche i due record `A` per i nomi `ns.bz.trl` e `ns.fi.trl` indicati come DNS per consentire la risoluzione del loro indirizzo IP. Se si fossero usati nomi relativi ad una zona diversa da quella delegata (ad esempio `nsfi.truelite.it` o `nsbz.truelite.it`) questo non sarebbe stato necessario. Una volta eseguita questa impostazione basterà configurare le due zone `fi.trl` e `bz.trl` sui server DNS indicati dai record `A`, ed il gioco è fatto.

9.3.2 La gestione di un secondario

Una delle caratteristiche più interessanti del DNS è quella che il protocollo prevede al suo interno il supporto per consentire in maniera naturale la distribuzione di carico e la ridondanza attraverso la possibilità di gestire dei *nameserver* secondari cui trasferire in maniera automatica tutte le informazioni di una certa zona.

Come accennato in sez. 9.2.4 è infatti sempre possibile indicare più di un server come *nameserver* per un dominio, e questi vengono automaticamente fatti girare nelle risposte, in modo che i client che effettuano le richieste ottengano tutte le volte come prima risposta un server diverso, cosa che consente di distribuire il carico fra gli stessi.

Perché il meccanismo funzioni però occorre che tutti i *nameserver* indicati siano coerenti fra loro, e per questo il protocollo mette a disposizione un meccanismo specifico, chiamato *trasferimento di zona*, che consente di sincronizzare i dati presenti su un server principale, detto *primario*, con tutti i server *secondari* che riceveranno automaticamente gli aggiornamenti.

Perché questo sia possibile occorre però configurare il *primario* per consentire i trasferimenti di zona, essendo infatti questa una operazione potenzialmente onerosa è normalmente opportuno abilitarla esplicitamente solo per i server che effettivamente si andranno ad utilizzare. Per questo se ad esempio vogliamo creare un secondario per `truelite.it`, dovremo avere nel `named.conf` del primario una dichiarazione del tipo:

named.conf
```
zone "truelite.it" {
        type master;
        file "/etc/bind/db.truelite.it";
        allow-transfer { 62.94.0.2; 62.48.33.131; 62.48.33.132; };
};
```

dove rispetto agli esempi di sez. 9.2.4 si è utilizzata l'ulteriore l'istruzione `allow-transfer` per indicare gli indirizzi IP dei *nameserver* secondari che saranno i soli autorizzati a ricevere il trasferimento di zona.

La configurazione di un server DNS per fare da secondario è anch'essa molto semplice, si tratta semplicemente di dichiarare una zona di tipo `slave`, inserendo in `named.conf` una voce analoga alla seguente:

named.conf
```
zone "truelite.it" {
        type slave;
        file "db.truelite.it";
        masters {
                62.48.34.25;
        };
};
```

In questo caso la zona viene dichiarata come di tipo `slave`, e l'istruzione `file` stavolta indica il file su cui verranno scritti i dati della zona direttamente da *BIND*, che li otterrà dal primario. L'istruzione `masters` consente di indicare una lista di server DNS primari da cui ottenere i dati della relativa zona.

Appendice A

Indice degli argomenti per LPI

A.1 Argomenti Certificazione LPIC-1

Questa sezione contiene gli indici degli argomenti relativi ai due esami (identificati dai numeri 101 e 102) necessari a raggiungere la certificazione LPI di primo livello (LPIC-1) "*Junior Level Linux Certification*". I riferimenti aggiornati agli argomenti della certificazione possono essere ottenuti su sito del Linux Professional Institute all'indirizzo: `http://www.lpi.org/linux-certifications/programs/lpic-1`.

A.1.1 Argomenti Esame LPI 101

Nella tabella seguente sono riportati i titoli e gli identificativi numerici degli obiettivi coperti dal primo dei due esami della certificazione LPIC-1 (LPI 101) associati ai corrispondenti riferimenti alle sezioni del testo in cui detti argomenti sono trattati.

Topic	**Titolo**	**Riferimento**
101	*System Architecture*	
101.1	Determine and configure hardware settings	sez. 5.1.1, 5.1.3, 5.2.5 e 5.4
101.2	Boot the system	sez. 1.1.2, 1.3.1 e 5.3.1
101.3	Change runlevels and shutdown or reboot system	sez. 5.3.5, 5.3.7, e 5.3.6
102	*Linux Installation and Package Management*	
102.1	Design hard disk layout	sez. 1.2.3 e 5.1.2
102.2	Install a boot manager	sez. 5.3.2 e 5.3.3
102.3	Manage shared libraries	sez. 3.1.2
102.4	Use Debian package management	sez. 4.2.3
102.5	Use RPM and YUM package management	sez. 4.2.2
103	*GNU and Unix Commands*	
103.1	Work on the command line	sez. 2.1.1, 2.1.3, 2.2.1
103.2	Process text streams using filters	sez. 2.2.3, 2.2.4, 2.2.5
103.3	Perform basic file management	sez. 1.2.2, 2.1.3, 2.2.2
103.4	Use streams, pipes and redirects	sez. 2.1.4 e 2.4.1
103.5	Create, monitor and kill processes	sez. 1.3 e 2.1.2
103.6	Modify process execution priorities	sez. 1.3.4
103.7	Search text files using regular expressions	sez. 2.2.5

103.8	Perform basic file editing operations using vi	sez. 2.3.3
104	*Devices, Linux Filesystems, Filesystem Hierarchy Standard*	
104.1	Create partitions and filesystems	sez. 5.1.2, 5.1.4 e 5.1.7
104.2	Maintain the integrity of filesystems	sez. 5.1.3 e 5.1.5
104.3	Control mounting and unmounting of filesystems	sez. 5.1.3
104.4	Manage disk quotas	sez. 6.4
104.5	Manage file permissions and ownership	sez. 1.4
104.6	Create and change hard and symbolic links	sez. 1.2.2
104.7	Find system files and place files in the correct location	sez. 1.2.3, 2.1.3 e 2.2.2.

A.1.2 Argomenti Esame LPI 102

Nella tabella seguente sono riportati i titoli e gli identificativi numerici degli obiettivi coperti dal secondo dei due esami della certificazione LPIC-1 (LPI 102) associati ai corrispondenti riferimenti alle sezioni del testo in cui detti argomenti sono trattati.

Topic	**Titolo**	**Riferimento**
105	*Shells, Scripting and Data Management*	
105.1	Customize and use the shell environment	sez. 2.1.3 e 2.1.6
105.2	Customize or write simple scripts	sez. 2.1.3 e 2.1.5
105.3	SQL data management	sez. 4.4
106	*User Interfaces and Desktops*	
106.1	Install and configure X11	sez. 3.3.1, 3.3.2
106.2	Setup a display manager	sez. 3.3.3
106.3	Accessibility	sez. 3.3.5
107	*Administrative Tasks*	
107.1	Manage user and group accounts and related system files	sez. 4.3.1, 4.3.2, 4.3.3
107.2	Automate system administration tasks by scheduling jobs	sez. 3.2.1
107.3	Localisation and internationalisation	sez. 3.1.4 e 2.1.3
108	*Essential System Services*	
108.1	Maintain system time	sez. 2.4.3 e 8.1.4
108.2	System logging	sez. 3.2.3 e 3.2.4
108.3	Mail Tranfer Agent (MTA) basics	sez. 8.5
108.4	Manage printers and printing	sez. 3.4
109	*Networking Fundamentals*	
109.1	Fundamentals of internet protocols	sez. 7.1 e 7.2
109.2	Basic network configuration	sez. 7.3
109.3	Basic network troubleshooting	sez. 7.6
109.4	Configure client side DNS	sez. 7.4
110	*Security*	
110.1	Perform security administration tasks	sez. 7.6.3, 4.3.5
110.2	Setup host security	sez. 8.1.5
110.3	Securing data with encryption	sez. 8.3 e 2.4.6

A.2 Argomenti Certificazione LPIC-2

Questa sezione contiene gli indici degli argomenti relativi ai due esami (identificati dai numeri 201 e 202) necessari a raggiungere la certificazione LPI di secondo livello (LPIC-2) "*Advanced*

Level Linux Certification". I riferimenti aggiornati agli argomenti possono essere ottenuti su sito del Linux Professional Institute all'indirizzo: `http://www.lpi.org/linux-certifications/programs/lpic-2`.

A.2.1 Argomenti Esame LPI 201

Nella tabella seguente sono riportati i titoli e gli identificativi numerici degli obiettivi coperti dal primo dei due esami della certificazione LPIC-2 (LPI 201) associati ai corrispondenti riferimenti alle sezioni del testo in cui detti argomenti sono trattati.

Topic	**Titolo**	**Riferimento**
201	*Linux Kernel*	
201.1	Kernel Components	sez. 5.2.1
201.2	Compiling a kernel	sez. 5.2.3 e 5.2.4
201.3	Patching a kernel	sez. 5.2.2
201.4	Customise, build and install a custom kernel and kernel modules	sez. 5.2.3
201.5	Manage/Query kernel and kernel modules at runtime	sez. 5.2.5
202	*System Startup*	
202.1	Customising system startup and boot processes	sez. 5.3.5 e 5.3.7
202.2	System recovery	sez. 5.1.5, 5.3.3 e 5.3.7
203	*Filesystem and Devices*	
203.1	Operating the Linux filesystem	sez. 5.1.3 e 5.1.7
203.2	Maintaining a Linux filesystem	sez. 5.1.4 e 5.1.5
203.3	Creating and configuring filesystem options	sez. 5.1.6 e 5.1.7
203.4	udev Device Management	sez. 5.4.5
204	*Advanced Storage Device Administration*	
204.1	Configuring RAID	sez. 6.1
204.2	Adjusting Storage Device Access	sez. 6.3
204.3	Logical Volume Manager	sez. 6.2
205	*Networking Configuration*	
205.1	Basic networking configuration	sez. 7.3, sez. 7.6, sez. 7.5.4, sez. 7.5.5
205.2	Advanced Network Configuration and Troubleshooting	sez. 7.3, sez. 7.6, sez. 7.7 e sez. 4.3 di [SGL]
205.3	Troubleshooting network issues	sez. 7.3, sez. 7.4, sez. 7.6, sez. 8.1.5, sez. 9.1.3
205.4	Notify users on system-related issues	sez. 5.3.7 e 4.3.5
206	*System Maintenance*	
206.1	Make and install programs from source	sez. 4.2.1
206.2	Backup operations	sez. 4.1
207	*Domain Name Server*	
207.1	Basic DNS server configuration	sez. 9.2
207.2	Create and maintain DNS zones	sez. 9.2.4, sez. 9.3
207.3	Securing a DNS server	non trattato

A.2.2 Argomenti Esame LPI 202

Nella tabella seguente sono riportati i titoli e gli identificativi numerici degli obiettivi coperti dal secondo dei due esami della certificazione LPIC-2 (LPI 202) associati ai corrispondenti riferimenti

alle sezioni del testo in cui detti argomenti sono trattati.

Topic	**Titolo**	**Riferimento**
208	*Web Services*	
208.1	Implementing a web server	non trattato
208.2	Maintaining a web server	sez. 2.1 di [SGL]
208.3	Implementing a proxy server	non trattato
209	*File Sharing*	
209.1	SAMBA Server Configuration	sez. 8.4.3 (solo cenni)
209.2	NFS Server Configuration	sez. 8.4.1
210	*Network Client Management*	
210.1	DHCP configuration	sez. 8.2.3
210.2	PAM authentication	sez. 4.3.7
210.3	LDAP client usage	sez. 1.2.5 di [IS-LDAP]
211	*E-Mail Services*	
211.1	Using e-mail servers	non trattato
211.2	Managing Local E-Mail Delivery	non trattato
211.3	Managing Remote E-Mail Delivery	non trattato
212	*System Security*	
212.1	Configuring a router	sez. 7.3.4, cap. 3 di [SGL]
212.2	Securing FTP servers	non trattato
212.3	Secure shell (SSH)	sez. 8.3
212.4	TCP Wrapper	sez. 8.1.5
212.5	Security tasks	sez. 5.2 e 5.4 di [SGL]
213	*Troubleshooting*	
213.1	Identifying boot stages and troubleshooting bootloaders	sez. 5.3
213.2	General troubleshooting	non trattato
213.3	Troubleshooting system resources	non trattato
213.4	Troubleshooting environment configurations	non trattato

A.3 Percorsi di studio per la certificazione LPI

Questa sezione presenta i percorsi di studio attraverso i capitoli del libro per coprire gli argomenti previsti dai vari esami richiesti per il raggiungimento delle certificazioni LPI.

A.3.1 Percorso di studio per l'esame LPI 101

Il percorso di studio consigliato per gli argomenti relativi all'esame LPI 101 è il seguente. Si tenga conto che il percorso non segue l'ordine degli argomenti elencati nelle pagine di descrizione degli obiettivi riportati negli indici di sez. A.1.1.

- Capitolo 1, tutto, tralasciare eventualmente la panoramica di sez. 1.1.
- Capitolo 2:
  - da sez. 2.1.1 a sez. 2.1.4,
  - sez. 2.2 tutta,
  - sez. 2.3.3,
  - sez. 2.4.1;

- Capitolo 3:
  - sez. 3.1.2, si legga eventualmente l'introduzione di sez. 3.1.1.
- Capitolo 4:
  - sez. 4.1.2,
  - sez. 4.2 tutta,
  - sez. 4.4.4;
- Capitolo 5:
  - sez. 5.1 tutta,
  - sez. 5.3 tutta,
  - sez. 5.4 tutta, tralasciare eventualmente i dettagli su 5.4.2, 5.4.3 e 5.4.4.

A.3.2 Percorso di studio per l'esame LPI 102

Il percorso di studio consigliato per gli argomenti relativi all'esame LPI 102 è il seguente. Si tenga conto che il percorso non segue l'ordine degli argomenti elencati nelle pagine di descrizione degli obiettivi riportati negli indici di sez. A.1.2.

- Capitolo 2:
  - ripassare sez. 2.1.3, ed eventualmente sez. 2.1.4,
  - approfondire su sez. 2.1.5 e sez. 2.1.6,
  - per la gestione dell'orario di sistema sez. 2.4.3,
  - per GPG la relativa parte di sez. 2.4.6.
- Capitolo 3:
  - tutto a partire da sez. 3.1.4, si legga eventualmente l'introduzione di sez. 3.1.1.
- Capitolo 4:
  - tutta sez. 4.3,
  - sez. 4.4.4.
- Capitolo 7:
  - tutto tranne, sez. 7.6.4, sez. 7.6.5, sez. 7.7.3 e sez. 7.5,
  - se pratici di reti e protocolli si può sorvolare su sez. 7.1 e sez. 7.2.
- Capitolo 8:
  - sez. 8.1,
  - sez. 8.3, eccetto sez. 8.3.4;
  - sez. 8.5.

Appendice B

Ringraziamenti

Si ringraziano coloro che han permesso la realizzazione di queste dispense contribuendo al finanziamento della loro scrittura:

- la Scuola per la Formazione Professionale Luigi Einaudi di Bolzano
- il Fondo Sociale Europeo nella realizzazione di corsi finanziati
- LiCI (`http://www.lici.it`) per il supporto e l'adozione come libro di testo.

Si ringraziano inoltre per l'aiuto nella stesura e nella revisione del testo:

- gli studenti del master "*Free Software Upgrade*" per il grande lavoro di revisione effettuato
- Gabriele Bertolucci, per la puntuale rilettura, i suggerimenti e la segnalazione di numerosi di errori sfuggiti a tutte le revisioni precedenti.

Appendice C

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

C.1 Applicability and Definitions

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "*Document*", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "*you*". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "*Modified Version*" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "*Secondary Section*" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "*Invariant Sections*" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "*Cover Texts*" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "*Transparent*" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "*Opaque*".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "*Title Page*" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For

works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "*publisher*" means any person or entity that distributes copies of the Document to the public.

A section "*Entitled XYZ*" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "*Acknowledgements*", "*Dedications*", "*Endorsements*", or "*History*".) To "*Preserve the Title*" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

C.2 Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

C.3 Copying in Quantity

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using

public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

C.4 Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You

may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

C.5 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

C.6 Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

C.7 Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

C.8 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

C.9 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

C.10 Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

C.11 Relicensing

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business

in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

C.12 Addendum: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Indice analitico

access point, 540–541
advanced routing, 522–525
arithmetic expansion, 80–81
autofs, 342–344

backup, 225–230
bootloader, 3, 380–381, 390
broadcast, 496–497

canale DMA, 414
canali DMA, 414
chroot, 52–53
CIDR, 496
comando
- `addgroup`, 267
- `adduser`, 267
- `apropos`, 142
- `apt-cache`, 256
- `apt-cdrom`, 253
- `apt-get`, 253
- `aptitude`, 255
- `arp`, 552
- `atq`, 185
- `atrm`, 185
- `at`, 184
- `batch`, 185
- `blkid`, 320
- `bunzip2`, 119
- `bzcat`, 119
- `bzip2`, 119
- `cancel`, 221
- `cat`, 84
- `cfdisk`, 311
- `cgdisk`, 314
- `chage`, 267
- `chattr`, 55
- `chat`, 536
- `chfn`, 268
- `chgpasswd`, 269
- `chgrp`, 52
- `chkconfig`, 398
- `chmod`, 51
- `chown`, 52
- `chpasswd`, 269
- `chroot`, 53
- `chsh`, 268
- `cksum`, 156
- `cpio`, 233
- `cp`, 15
- `createdb`, 297
- `createuser`, 297
- `crontab`, 183
- `cryptsetup`, 472
- `cupsaccept`, 220
- `cupsdisable`, 220
- `cupsenable`, 220
- `cupsreject`, 220
- `cut`, 114
- `date`, 146
- `dd`, 237
- `debugfs`, 340
- `delgroup`, 267
- `deluser`, 267
- `depmod`, 375
- `df`, 326
- `dhclient`, 580
- `diff`, 351
- `dig`, 626
- `dmesg`, 152
- `dpkg-reconfigure`, 258
- `dpkg`, 250
- `dropdb`, 297

dropuser, 297
dump, 235
du, 327
e2fsck, 337
e2image, 340
echo, 72
edquota, 479
env, 75
expand, 118
exportfs, 603
false, 96
fdisk, 309
findfs, 320
find, 106
finger, 559
fmt, 120
fold, 120
free, 35
fsck.xfs, 338
fsck, 336
ftp, 558
fuser, 153
gdisk, 313
genisoimage, 347
getent, 283
getty, 61
gpart, 315
gpasswd, 269
gpg, 159
grep, 120
groupadd, 266
groupmod, 266
groups, 149
grub-install, 384
grub-mkconfig, 387
grub, 384
gunzip, 119
gzip, 118
halt, 411
hdparm, 462
hd, 112
head, 113
hexdump, 111
hostname, 532
host, 626
hwclock, 148
iconv, 178
id, 149
ifconfig, 504
ifdown, 518
ifup, 518
info, 144
insmod, 371
ip, 519
iwconfig, 542
iwlist, 543
join, 115
journalctl, 409
killall, 39
kill, 37
lastb, 151
last, 151
ldconfig, 172
ldd, 171
less, 111
ln, 12
locale-gen, 177
locale, 176
locate, 105
logger, 190
login, 61
logname, 150
logrotate, 190
losetup, 322
lpadmin, 219
lpinfo, 222
lpoptions, 222
lpq, 224
lprm, 224
lpr, 223
lpstat, 222
lp, 221
lsattr, 54
lsmod, 377
lsof, 152
lspci, 417
lsscsi, 423
lsusb, 429
ls, 9
lvcreate, 460

lvdisplay, 461
lvextend, 461
lvreduce, 461
lvremove, 461
lvresize, 461
lvscan, 461
lzma, 119
mailq, 619
mailx, 616
mail, 616
make, 242
mandb, 143
manpath, 143
man, 141
md5sum, 156
mdadm, 443
minicom, 426
mkdir, 17
mke2fs, 329
mkfifo, 54
mkfontdir, 198
mkfs.vfat, 327
mkfs.xfs, 333
mkfs, 327
mkinitramfs, 369
mkinitrd, 368
mkisofs, 347
mknod, 53
mkreiserfs, 327
mkswap, 345
modinfo, 376
modprobe, 371
more, 110
mount.cifs, 612
mount, 317
mtr, 548
mv, 14
mysqladmin, 301
mysqldump, 301
mysql, 299
nc, 556
netcat, 556
netstat, 549
net, 610
newaliases, 618
newgrp, 271
nice, 40
nl, 120
nohup, 65
nologin, 280
ntpdate, 148
ntpq, 574
od, 111
parted, 470
passwd, 267
paste, 114
patch, 352
pg_dumpall, 298
pg_dump, 297
pgrep, 39
pidof, 36
ping, 544
pkill, 39
poweroff, 411
pr, 120
psql, 296
pstree, 25
ps, 26
pump, 580
pvcreate, 455
pvdisplay, 456
pvmove, 457
pvresize, 457
pvscan, 455
pwd, 17
quotacheck, 478
quotaoff, 479
quotaon, 478
quotatool, 482
quota, 480
reboot, 411
reiserfsck, 338
renice, 41
repquota, 480
resize2fs, 469
resize_reiserfs, 469
restore, 236
rmdir, 17
rmmod, 376
rm, 14

rndc, 629
route, 508
rpcinfo, 554
rpm2cpio, 244
rpm, 244
rsync, 239
run-parts, 182
scp, 593
screen, 66
scsiinfo, 423
sdparm, 465
sed, 124
sendmail, 619
seq, 96
service, 411
setquota, 481
setserial, 424
sfdisk, 311
sgdisk, 314
sg, 271
shasum, 157
shred, 14
shutdown, 412
smbclient, 611
smbmount, 612
smbpasswd, 610
smbstatus, 612
sort, 116
split, 113
ssh-add, 597
ssh-copy-id, 596
ssh-keygen, 595
ssh, 591
startx, 203
strace, 154
strings, 155
sudoedit, 272
sudo, 271
su, 270
swapoff, 346
swapon, 346
sync, 326
sysctl, 173
syslinux, 382
systemctl, 407
tac, 112
tail, 113
tar, 230
tcpdchk, 576
tcpdmatch, 577
tcpd, 574
tee, 140
telinit, 399
telnet, 555
testparm, 608
test, 96
time, 146
top, 33
touch, 10
tracepath, 546
traceroute, 545
tree, 18
true, 96
tr, 117
ttmkfontdir, 198
tune2fs, 331
tzselect, 179
udevadm, 434
udevinfo, 434
udevmonitor, 434
umount, 325
uname, 151
unexpand, 118
uniq, 117
unxz, 120
unzip, 118
update-grub, 385
update-initramfs, 369
update-rc.d, 398
updatedb, 105
uptime, 151
useradd, 264
usermod, 266
users, 150
vgchange, 458
vgcreate, 457
vgdisplay, 459
vgextend, 458
vgmerge, 459
vgreduce, 458

vgscan, 459
vgsplit, 460
visudo, 276
wall, 412
wc, 112
whatis, 142
whereis, 105
which, 77
whoami, 149
whois, 560
who, 150
w, 150
xargs, 139
xauth, 209
xfs_admin, 335
xfs_db, 341
xfs_fsr, 342
xfs_growfs, 469
xfs_info, 341
xfs_metadump, 341
xfs_repair, 338
xhost, 209
xinit, 202
xrdb, 210
xterm, 203
xzcat, 120
xz, 119
yaird, 369
yes, 141
yumdownloader, 248
yum, 247
zcat, 119
built-in
  alias, 78
  bg, 64
  cd, 17
  declare, 74
  disown, 65
  exec, 95
  exit, 65
  export, 74
  fg, 64
  history, 81
  jobs, 64
  read, 92
  set, 101
  source, 95
  typeset, 74
  type, 78
  ulimit, 289
  umask, 50
  unalias, 78
  unset, 72
  wait, 64
command expansion, 80
configurazione
  .Xresources, 210
  .bash_login, 99
  .bash_logout, 100
  .bash_profile, 99
  .bashrc, 100
  .fonts.conf, 198
  .profile, 99
  .xinitrc, 203
  /boot/grub/grub.cfg, 387
  /etc/X11/xinit/xinitrc, 203
  /etc/X11/xorg.conf, 195
  /etc/anacrontab, 182
  /etc/apt/sources.list.d/, 251
  /etc/apt/sources.list, 251
  /etc/auto.master, 342
  /etc/bash.bashrc, 100
  /etc/bash_completion, 83
  /etc/bootptab, 578
  /etc/cron.allow, 183
  /etc/cron.deny, 184
  /etc/crontab, 180
  /etc/cups/cupsd.conf, 216
  /etc/default/autofs, 344
  /etc/default/grub, 387
  /etc/dhcp/dhclient.conf, 581
  /etc/dhcp/dhcpd.conf, 583
  /etc/ethers, 553
  /etc/exports, 601
  /etc/filesystems, 318
  /etc/fonts/fonts.conf, 198
  /etc/fonts/local.conf, 198
  /etc/fstab, 322
  /etc/gdm/gdm.conf, 205
  /etc/group, 260

/etc/gshadow, 264
/etc/hdparm.conf, 464
/etc/host.conf, 530
/etc/hosts.allow, 575
/etc/hosts.deny, 575
/etc/hosts, 531
/etc/inetd.conf, 564
/etc/initramfs-tools/initramfs.conf, 369
/etc/inittab, 394
/etc/iproute2/rt_tables, 522
/etc/issue.net, 277
/etc/issue, 277
/etc/kde4/kdm/kdmrc, 205
/etc/ld.so.cache, 172
/etc/ld.so.conf, 172
/etc/lightdm/lightdm.conf, 207
/etc/localtime, 179
/etc/login.defs, 278
/etc/logrotate.conf, 192
/etc/lvm/lvm.conf, 455
/etc/manpath.config, 143
/etc/mdadm/mdadm.conf, 444
/etc/mke2fs.conf, 330
/etc/modprobe.conf, 373
/etc/modprobe.d/, 373
/etc/modules, 377
/etc/motd, 277
/etc/mtab, 325
/etc/mysql/my.cnf, 299
/etc/network/interfaces, 515
/etc/networks, 533
/etc/nsswitch.conf, 282
/etc/ntp.conf, 573
/etc/pam.conf, 285
/etc/pam.d, 284
/etc/passwd, 260
/etc/ppp/options, 535
/etc/ppp/peers, 536
/etc/profile, 98
/etc/protocols, 534
/etc/rc.local, 398
/etc/resolv.conf, 528
/etc/rpc, 554
/etc/rsyslog.conf, 186
/etc/samba/smb.conf, 605
/etc/securetty, 278
/etc/security/limits.conf, 290
/etc/services, 532
/etc/shadow, 263
/etc/shells, 280
/etc/skel, 280
/etc/ssh/ssh_config, 589
/etc/ssh/sshd_config, 586
/etc/ssh/sshrc, 592
/etc/sudoers, 272
/etc/sysctl.conf, 175
/etc/sysctl.d/, 175
/etc/syslog.conf, 186
/etc/systemd/journald.conf, 409
/etc/systemd/system.conf, 403
/etc/systemd/system/, 404
/etc/systemd/user.conf, 403
/etc/udev/udev.conf, 432
/etc/updatedb.conf, 106
/etc/xinetd.conf, 567
/etc/yum.conf, 248
/etc/yum.repos.d/, 249
menu.lst, 384
named.conf, 629
pg_hba.conf, 295
postgresql.conf, 294

default gateway, 500–501
demone
anacron, 182
atd, 184
bootpd, 578
cron, 180
cupsd, 215
dhcpd, 583
gdm, 204
inetd, 564
journald, 409
kdm, 205
klogd, 186
lightdm, 206
mysqld, 299
named, 628
nmbd, 605
ntpd, 572

portmap, 554
postmaster, 294
pppd, 535
rsyslogd, 186
smbd, 605
ssh-agent, 597
sshd, 586
syslogd, 185
systemd, 403
udevd, 432
xdm, 204
xfs, 197
xinetd, 567
device mapper, 460, 472
directory di lavoro, 16–17
directory radice, 18–25
disciplina di linea, 556

editor
emacs, 128
jed, 135
joe, 135
nano, 137
pico, 137
vi, 131
El Torito, 347
espressioni regolari, 120–124
ESSID, 541

fifo, 7, 54
filename globbing, 78–80
filesystem /proc
/proc/bus/pci, 417
/proc/bus/usb, 428
/proc/cmdline, 392
/proc/dma, 415
/proc/filesystems, 318
/proc/interrupts, 415
/proc/ioports, 415
/proc/mdstat, 447
/proc/modules, 377
/proc/mounts, 325
/proc/partitions, 319
/proc/swaps, 346
/proc/sys/kernel/hotplug, 430
/proc/sys/kernel/modprobe, 370
/proc/sys/net/ipv4/ip_forward, 511
Filesystem Hierarchy Standard, 19–25
filesystem temporaneo, 21, 432
file di dispositivo, 7, 53–54
firewall mark, 524
Fully Qualified Domain Name, 621

greeter, 204
gruppo ausiliario, 43, 261
gruppo primario, 42–43, 261
gruppo principale, 42–43, 261

hard link, 12–13
hash, 156
history, 81–82
history expansion, 81
hostname, 531–532
hotplug, 430–431

inode, 11–15, 329–330
interrupt, 414
IP aliasing, 506

Joliet, 347
journaling, 336–337

kernel boot parameters, 391–392
kernel panic, 355
keyfile, 169–170

link-loader, 89, 170–171
link-local address, 499
Logical Volume Manager, 452–462
login manager, 204

MAC address, 492
Mail Transport Agent, 613
Mail User Agent, 613
major number, 53
Master Boot Record, 309
memoria virtuale, 3
minor number, 53
modo promiscuo, 507
mount point, 317
multicast, 495–496

Name Service Switch, 281–283
nice, 40–41
nome a dominio, 526–529

pager, 110
page fault, 345
pathname, 16–17
path search, 76
pipelining, 87–88
Pluggable Authentication Modules, 284–288
Point to Point Protocol, 534–535
porte, 501–503
  destinazione, 502
  effimere, 503
  riservate, 503
  sorgente, 502
prompt, 69

ramdisk, 368–370, 392–393
Remote Procedure Call, 553–555
resolver, 526–527
Resource Record, 624
Rock Ridge, 347
router, 500–501
routing, 499–501, 508–513
runlevel, 394–413

sessione di lavoro, 60
sgid, 47–49
shadow password, 262–264, 267–268
shell
  `ash`, 59
  `bash`, 59
  `csh`, 59
  `ksh`, 59
  `sh`, 59
  `tcsh`, 59
  `zsh`, 59
  di login, 97–100, 280
  interattiva, 100–101
  istruzione
    `break`, 94
    `case`, 93
    `continue`, 94
    `done`, 93
    `do`, 93
    `elif`, 93
    `else`, 92
    `esac`, 93
    `fi`, 92
    `for`, 94
    `function`, 94
    `if`, 92
    `then`, 92
    `until`, 93
    `while`, 93
  restricted, 280
  ristretta, 101
  subshell, 95–96
single user mode, 394
socket, 490–491
socket activation, 401–402
standard error, 60–61, 83–88
standard input, 60–61, 83–88
standard output, 60–61, 83–88
sticky, 48–49
suid, 47–49
swap, 344–347
sysfs, 431–432
syslog, 185–193
system call, 2

tabella di instradamento, 508–509
terminale di controllo, 60–65
timezone, 178–179
TOS, 524

udev, 430–434
UEFI, 379–381
umask, 50
unit file, 403–404
Universal Unique Identifier, 319–320

variabili
  di ambiente, 73–75
    `DISPLAY`, 208
    `EDITOR`, 75
    `HOME`, 75
    `LANG`, 75–76
    `LD_LIBRARY_PATH`, 173

LD_PRELOAD, 173
LPDEST, 224
MANPATH, 143
PAGER, 110
PATH, 76–77
PRINTER, 223
SSH_AUTH_SOCK, 597
TAPE, 231
TERM, 75
TZ, 179
USER, 73
XAUTHORITY, 209
di shell, 70–75
PS2, 70
BASH, 99
HISTFILESIZE, 81
HISTFILE, 81
HISTSIZE, 81
IFS, 75
PS1, 70
speciali, 91
variable expansion, 71–72

window manager, 211

XML, 198

zombie, 29–31
zona di autorità, 624–625

Bibliografia

[AdvBash] Mendel Cooper. *Advanced Bash-Scripting Guide.* http://www.tldp.org/LDP/abs/, 2000.

[DebSys] Martin F. Krafft. *Debian system, concepts and tecniques.* William Pollok, 2005.

[GaPiL] Simone Piccardi. *Guida alla Programmazione in Linux.* http://gapil.gnulinux.it, 2012.

[IS-LDAP] Simone Piccardi. *Integrazione sistemistica con LDAP.* http://labs.truelite.it, 2011.

[Kerberos] Jason Garman. *Kerberos: The Definitive Guide.* O'Reilly, 2003.

[LinuxNutshell] Greg Kroah-Hartman. *Linux kernel in a Nutshell.* O'Reilly, 2006.

[Make] Paul D. Smith Richard M. Stallman, Roland McGrath. *GNU make.* Free Software Foundation, 2004.

[RegExp] Jeffrey E. F. Friedl. *Mastering regular expression.* O'Reilly, 1997.

[SGL] Simone Piccardi. *La sicurezza con GNU/Linux.* http://labs.truelite.it, 2003.

[TCPIll1] W. Richard Stevens. *TCP/IP Illustrated, Volume 1, the protocols.* Addison Wesley, 1994.

[WebServ] Simone Piccardi. *I servizi web.* http://labs.truelite.it, 2003.

Amministrare GNU/Linux

Amministrare GNU/Linux è un testo introduttivo alla amministrazione di sistema in ambiente GNU/Linux. Scopo del testo è trattare gli argomenti di base ed avanzati dell'amministrazione di sistema ed i principi di base dell'amministrazione di rete.

Si è adottato un percorso espositivo che parte dall'esame della struttura del sistema per arrivare ad affrontare gli argomenti più sofisticati e complessi della gestione sistemistica, senza dare per presupposta nessuna conoscenza di questo sistema operativo.

Il testo copre anche gli argomenti previsti negli esami di certificazione LPI 101 e 102, ed una buona parte degli argomenti dei successivi esami 201 e 202.

Può pertanto essere utilizzato come testo (non ufficiale) di studio per la preparazione di detti esami, e per questo è stato scelto come libro di testo da Linux Certification Institute (www.lici.it) e da molti altri centri di formazione.

Restando fedele alla filosofia di sviluppo di GNU/Linux il testo è pubblicato con licenza libera, e può essere scaricato gratuitamente.

Simone Piccardi, laureato in fisica, ha partecipato a numerosi progetti di ricerca nel campo dell'astrofisica delle particelle, conseguendo il dottorato di ricerca. Da oltre 15 anni si interessa di software libero ed opera nel settore come professionista dal 2000.

Autore, oltre che del presente volume, di "Integrazione Sistemistica con LDAP" e numerosa documentazione libera, nel 2003 ha fondato Truelite Srl, una azienda specializzata nella fornitura di supporto professionale al software libero, di cui è responsabile tecnico.





LICI.IT